EPTCS 439

Proceedings of the

41st International Conference on

Logic Programming

Rende, Italy, 12-19th September 2025

Edited by: Martin Gebser, Daniela Inclezan, Francesco Ricca, Manuel Carro and Miroslaw Truszczynski

Published: 8th January 2026
DOI: 10.4204/EPTCS.439
ISSN: 2075-2180
Open Publishing Association

Introduction to the Technical Communications of the 41st International Conference on Logic Programming

Martin Gebser
University of Klagenfurt
martin.gebser@aau.at

Daniela Inclezan
Miami University
inclezd@miamioh.edu

Francesco Ricca
University of Calabria
francesco.ricca@unical.it

Manuel Carro
IMDEA Software Institute & Universidad Politécnica de Madrid
manuel.carro@imdea.org

Miroslaw Truszczynski
University of Kentucky
mirek@cs.uky.edu

Since the first conference in Marseille in 1982, the International Conference on Logic Programming (ICLP) has been the premier international event for presenting research in logic programming. These proceedings include the Technical Communications of the 41st ICLP, held on 12-19 September 2025 at the University of Calabria in Rende, Italy. The papers and extended abstracts in this volume address the following areas and topics: theoretical foundations, language design and programming methodologies, program analysis and optimization, applications and implementation methodologies. This volume features contributions to three submission tracks of ICLP 2025: the Main track, IJCAI fast track, and Recently Published Research track.

The ICLP 2025 Technical Communications include two types of contributions: regular papers and extended abstracts. Four extended abstracts, which were contributed to the Recently Published Research track, describe published work that has not yet been presented at ICLP or other main conferences related to logic programming (such as LPNMR, LOPSTR, or LPAR). The 36 contributions, including 33 regular papers and three extended abstracts, from the Main track and the IJCAI fast track present original research. Differently from the part of the ICLP 2025 proceedings published in a special issue of *Theory and Practice of Logic Programming*, these contributions have been directly submitted to the Technical Communications, comparable to short (regular) papers, or were considered as high-quality, yet still work in progress by the reviewers. In fact, the impact of ICLP Technical Communications is witnessed, e.g., by the *10 Years Test of Time Award* of ICLP 2024 for the Technical Communications paper "*Clingo = ASP + Control: Preliminary Report*". All contributions to the ICLP 2025 Technical Communications have been evaluated by three reviewers and were presented during ICLP 2025. This volume thematically groups the Technical Communications according to the sessions of the ICLP 2025 program.

Acknowledgements

We would like to begin by thanking everyone who helped organizing and making ICLP 2025 possible:

- Organizing Chairs:

  Carmine Dodaro, University of Calabria
  Antonio Ielo, University of Calabria
  Giuseppe Mazzotta, University of Calabria

- Administrative Chair:

  Pierfrancesco Pantusa, University of Calabria

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. i–vi, doi:10.4204/EPTCS.439.0

© Gebser, Inclezan, Ricca, Carro & Truszczynski
This work is licensed under the
Creative Commons Attribution License.

- Workshop Chairs:
  - Pierangela Bruno, University of Calabria
  - Jorge Fandinno, University of Nebraska at Omaha
- Doctoral Consortium Chairs:
  - Alice Tarzariol, University of Klagenfurt
  - Markus Hecher, CNRS, Artois University
- Autumn School Chairs:
  - Sarah Gaggl, TU Dresden
  - Manuel Hermenegildo, IMDEA Software Institute & Universidad Politécnica de Madrid
- Programming Contest Chairs:
  - Mario Alviano, University of Calabria
  - Wolfgang Faber, University of Klagenfurt
- Publicity Chairs:
  - Manuel Alejandro Borroto Santana, University of Calabria
  - Francesco Calimeri, University of Calabria

The ICLP 2025 Program Committee consisted of 79 representatives of logic programming groups in Australia, Austria, Belgium, Canada, China, Finland, France, Germany, Greece, Israel, Italy, Japan, Portugal, Singapore, Spain, Sweden, Turkey, the United Kingdom, and the United States:

Salvador Abreu, NOVA-LINCS / University of Evora

Mario Alviano, University of Calabria

Nicos Angelopoulos, The Pirbright Institute

Marcello Balduccini, Saint Joseph's University

Mutsunori Banbara, Nagoya University

Elena Bellodi, University of Ferrara

Stefano Bistarelli, University of Perugia

Bart Bogaerts, KU Leuven

Pedro Cabalar, University of Corunna

Roberta Calegari, Alma Mater Studiorum–University of Bologna

Angelos Charalambidis, Harokopio University

Michael Codish, Ben-Gurion University of the Negev

Stefania Costantini, University of L'Aquila

Alessandro Dal Palù, University of Parma

Marina De Vos, University of Bath

Marc Denecker, KU Leuven

Carmine Dodaro, University of Calabria

Thomas Eiter, TU Wien
Esra Erdem, Sabanci University
Wolfgang Faber, University of Klagenfurt
François Fages, INRIA Saclay
Jorge Fandinno, University of Nebraska at Omaha
Johannes K. Fichte, Linköping University
Fabio Fioravanti, University of Chieti-Pescara
Paul Fodor, Stony Brook University
Andrea Formisano, University of Udine
Gerhard Friedrich, University of Klagenfurt
Sarah Alice Gaggl, TU Dresden
Marco Gavanelli, University of Ferrara
Laura Giordano, DISIT, University of Eastern Piedmont
Ricardo Gonçalves, NOVA University Lisbon
Gopal Gupta, The University of Texas at Dallas
Carito Guziolowski, Centrale Nantes
Markus Hecher, CNRS, Artois University
Giovambattista Ianni, University of Calabria
Tomi Janhunen, Tampere University
Nikos Katzouris, NCSR Demokritos
Gabriele Kern-Isberner, TU Dortmund
Matthias Knorr, NOVA University Lisbon
Isabelle Kuhlmann, University of Hagen
Joao Leite, NOVA University Lisbon
Michael Leuschel, Heinrich-Heine University of Düsseldorf
Yuliya Lierler, University of Nebraska at Omaha
Vladimir Lifschitz, The University of Texas at Austin
Francesca Alessandra Lisi, University of Bari Aldo Moro
Yanhong A. Liu, Stony Brook University
Marco Maratea, University of Calabria
Viviana Mascardi, DIBRIS, University of Genova
Jose F. Morales, IMDEA Software Institute & Universidad Politécnica de Madrid
Johannes Oetsch, Jönköping University
Simona Perri, University of Calabria

Enrico Pontelli, New Mexico State University
Fabrizio Riguzzi, University of Ferrara
Ricardo Rocha, University of Porto
Chiaki Sakama, Wakayama University
Zeynep G. Saribatur, TU Wien
Torsten Schaub, University of Potsdam
Konstantin Schekotihin, University of Klagenfurt
Tom Schrijvers, KU Leuven
Dietmar Seipel, University of Würzburg
Theresa Swift, Johns Hopkins Applied Physics Lab
Paul Tarau, University of North Texas
Hans Tompits, TU Wien
Tran Cao Son, New Mexico State University
Miroslaw Truszczynski, University of Kentucky
Athénaïs Vaginay, University of Caen Normandy
Concepcion Vidal, University of Corunna
Germán Vidal, VRAIN, Polytechnic University of Valencia
Alicia Villanueva, VRAIN, Polytechnic University of Valencia
Johannes P. Wallner, TU Graz
Kewen Wang, Griffith University
David S. Warren, Stony Brook University
Felix Weitkämper, German UDS Potsdam
Stefan Woltran, TU Wien
Roland Yap, National University of Singapore
Jia-Huai You, University of Alberta
Zhizheng Zhang, Southeast University
Yuanlin Zhang, Texas Tech University
Neng-Fa Zhou, CUNY Brooklyn College and Graduate Center

We are grateful to the external reviewers for their great help in evaluating and discussing submissions:
Iosif Apostolakis
Joaquín Arias
João Barbosa
Alexander Beiser

Alessandro Bertagnon
Jasmin Blanchette
Aysu Bogatarkan
Elisa Böhl
Alessia Donata Camarda
Marco Ciccalè
Marco Cuccarini
Fabio Aurelio D'Asaro
Huimin Dong
Abeer Dyoub
Müge Fidan
Thom Frühwirth
Tobias Geibinger
Gregory Gelfond
Raúl Gutiérrez
Susana Hahn
Zachary Hansen
Nelson Higuera Ruiz
Ethan Hong
Spencer Killen
Riley Kinahan
Luca Laboccetta
Michael Morak
Ryan Kepler James Murphy
Hidetomo Nabeshima
Dominik Pichler
Samuele Pollaci
Abhiramon Rajasekharan
Sascha Rechenberger
Luis Angel Rodriguez Reiners
Masoud Sadrnezhaad
Takehide Soh
Davide Soldà
Klaus Strauch

Carlo Taticchi

Van-Giang Trinh

Robbe Van den Eede

Huaduo Wang

Yisong Wang

Daniel Weidner

Baturay Yılmaz

We gratefully acknowledge the University of Calabria for hosting the conference on its campus. We also thank the Artificial Intelligence Journal (AIJ), the Italian Association for Artificial Intelligence (AIxIA), the Gruppo Ricercatori e Utenti Logic Programming (GULP), the US National Science Foundation (NSF), and the University of Calabria for their generous support of the student travel grants.

We extend our sincere gratitude to Enrico Pontelli, President of the Association of Logic Programming (ALP), Marco Gavanelli, ALP Conference Coordinator, and all members of the ALP Executive Committee. We gratefully acknowledge Miroslaw Truszczynski, Editor-in-Chief of TPLP, and the staff at Cambridge University Press for their valuable support. The EPTCS Editorial Board and, in particular, Rob van Glabbeek provided extraordinary help to publish the Technical Communications. Our deep appreciation also goes to the Program Committee members, external reviewers, and the entire ICLP 2025 organizing committee for their tireless and exemplary contributions. Without you, this conference would not have been possible. We are especially thankful to all authors who submitted to ICLP 2025. Thank you for choosing this venue to share your ideas, hard work, and expertise.

Table of Contents

Preface i
Martin Gebser, Daniela Inclezan, Francesco Ricca, Manuel Carro and Miroslaw Truszczynski

Table of Contents vii

SCIFFprob: A Probabilistic Abductive Framework for Compliance Verification 1
Michela Vespa, Elena Bellodi, Marco Gavanelli and Evelina Lamma

Propositional Abduction via Only-Knowing: A Non-Monotonic Approach 5
Sanderson Molick and Vaishak Belle

An ASP-Based Framework for MUSes 18
Mohimenul Kabir and Kuldeep S Meel

Machine Learning Model Integration with Open World Temporal Logic for Process Automation . . . 34
Dyuman Aditya, Colton Payne, Mario Leiva and Paulo Shakarian

VERUS-LM: a Versatile Framework for Combining LLMs with Symbolic Reasoning 47
Benjamin Callewaert, Simon Vandevelde and Joost Vennekens

An LLM + ASP Workflow for Joint Entity-Relation Extraction 63
Trang Tran, Trung Hoang Le, Huiping Cao and Tran Cao Son

VECSR: Virtually Embodied Common Sense Reasoning System 76
Alexis R. Tudor, Joaquín Arias and Gopal Gupta

Logic Tensor Network-Enhanced Generative Adversarial Network 89
Nijesh Upreti and Vaishak Belle

Interpretable Hybrid Machine Learning Models Using FOLD-R++ and Answer Set Programming . 114
Sanne Wielinga and Jesse Heyninck

Correcting Autonomous Driving Object Detection Misclassifications with Automated Commonsense Reasoning 128
Keegan Kimbrell, Wang Tianhao, Feng Chen and Gopal Gupta

Computing Universal Plans for Partially Observable Multi-Agent Routing Using Answer Set Programming 143
Fengming Zhu and Fangzhen Lin

The Combined Problem of Online Task Assignment and Lifelong Path Finding in Logistics Warehouses: Rule-Based Systems Matter 167
Fengming Zhu, Weijia Xu, Yifei Guo and Fangzhen Lin

A framework for Conditional Reasoning in Answer Set Programming 188
Mario Alviano, Laura Giordano and Daniele Theseider Dupré

Enhancing Compilation-based ASP through Lazy Atom Discovery 202
Andrea Cuteri, Giuseppe Mazzotta and Francesco Ricca

Defeasible Conditionals using Answer Set Programming 206
Racquel Dennison, Jesse Heyninck and Thomas Meyer

Extracting Policies from Quantified Answer Set Programs 224
Martín Diéguez and Igor Stéphan

A Simple and Effective ASP-Based Tool for Enumerating Minimal Hitting Sets 238
Mohimenul Kabir and Kuldeep S Meel

Logic Programming with Extensible Types 248
Ivan Perez and Angel Herranz

Fixpoint Semantics for DatalogMTL with Negation 263
Samuele Pollaci

A Refined Operational Semantics for FreeCHR 278
Sascha Rechenberger and Thom Frühwirth

On the Trap Space Semantics of Normal Logic Programs 294
Van-Giang Trinh, Sylvain Soliman, François Fages and Belaid Benhamou

Reasoning about Medical Triage Optimization with Logic Programming 320
Jaikrishna Manojkumar Patil, Adam Chapman, Richard Knuszka, John Chapman and Paulo Shakarian

The ASP-based Nurse Scheduling System at the University of Yamanashi Hospital 334
Hidetomo Nabeshima, Mutsunori Banbara, Torsten Schaub and Takehide Soh

Hybrid MKNF for Aeronautics Applications: Usage and Heuristics 349
Arun Raveendran Nair Sheela, Florence De Grancey, Christophe Rey and Victor Charpenay

An ASP-based Solution to the Medical Appointment Scheduling Problem 367
Alina Vozna, Andrea Monaldini, Stefania Costantini, Valentina Pitoni and Dawid Pado

FÆRDXEL: An Expert System for Danish Traffic Law 383
Luís Cruz-Filipe and Jonas Vistrup

Explainable AI for Sperm Morphology: Integrating YOLO with FastLAS 397
Talissa Dreossi, Agostino Dovier, Francesca Corte Pause, Susy Urli and Martina Crociati

XAI-LAW: A Logic Programming Tool for Modeling, Explaining, and Learning Legal Decisions . . 405
Agostino Dovier, Talissa Dreossi, Andrea Formisano and Benedetta Strizzolo

Formally Explaining Decision Tree Models with Answer Set Programming 420
Akihiro Takemura, Masayuki Otani and Katsumi Inoue

xDNN(ASP): Explanation Generation System for Deep Neural Networks powered by Answer Set Programming 438
Ly Ly Trieu and Tran Cao Son

Implementing the First-Order Logic of Here and There 453
Jens Otten and Torsten Schaub

Automated Theorem Proving for Prolog Verification 469
Fred Mesnard, Thierry Marianne and Étienne Payet

Investigating the Grounding Bottleneck for a Large-Scale Configuration Problem: Existing Tools and Constraint-Aware Guessing 482
Veronika Semmelrock and Gerhard Friedrich

On Zeno-like Behaviors in the Event Calculus with Goal-directed Answer Set Programming 496
Ondřej Vašíček, Joaquin Arias, Jan Fiedor, Gopal Gupta, Brendan Hall, Bohuslav Křena, Brian Larson and Tomáš Vojnar

Inductive First-Order Formula Synthesis by ASP: A Case Study in Invariant Inference 511
Ziyi Yang, George Pîrlea and Ilya Sergey

Picat Through the Lens of Advent of Code 528
Neng-Fa Zhou, Cristian Grozea, Håkan Kjellerstrand and Oisín Mac Fhearaí

Most Probable Explanation in Probabilistic Answer Set Programming 542
Damiano Azzolini, Giuseppe Mazzotta, Francesco Ricca and Fabrizio Riguzzi

ASPECT: Answer Set rePresentation as vEctor graphiCs in laTex 546
Alessandro Bertagnon and Marco Gavanelli

Probabilistic Traces in Declarative Process Mining 549
Michela Vespa, Elena Bellodi, Federico Chesani, Daniela Loreti, Paola Mello, Evelina Lamma, Anna Ciampolini, Marco Gavanelli and Riccardo Zese

Discovery of Logic-probabilistic Rules from COVID-19 Vaccine Antibody Response in Older People: Results from the GeroCovid VAX Study 552
Michela Vespa, Francesca Remelli, Elena Bellodi and Raffaele Antonelli Incalzi

$\mathscr{S}$CIFF^{Prob}: A Probabilistic Abductive Framework for Compliance Verification

Michela Vespa
michela.vespa@unife.it

Elena Bellodi
elena.bellodi@unife.it

Marco Gavanelli
marco.gavanelli@unife.it

Evelina Lamma
evelina.lamma@unife.it

Department of Engineering,
University of Ferrara
Via Saragat 1, Ferrara, Italy

Abductive Logic Programming (ALP) is a widely studied research direction in LP, and a variety of proof-procedures freely available makes it an appealing language to address complex challenges. In order to address also uncertainty and probabilistic reasoning, the proof procedure IFF^{Prob} enables probabilistic abduction with variables in ALP programs through integrity constraints (ICs) annotated with probabilities. On a separate branch, the $\mathscr{S}$CIFF proof-procedure supports run-time verification by handling dynamically incoming events. We introduce $\mathscr{S}$CIFF^{Prob}, a new proof procedure that integrates probabilistic ICs with dynamic event streams, and we demonstrate its application to probabilistic compliance in Process Mining.

1 Introduction

Abductive Logic Programming (ALP) [8] provides a declarative paradigm for reasoning with incomplete information, using abduction to generate hypotheses that explain observed facts. The $\mathscr{S}$CIFF framework [3, 4] extends ALP with explicit temporal expectations, enabling runtime monitoring and verification of agent compliance to interaction protocols in dynamic systems. Relations among events and expectations are formalized through Integrity Constraints (ICs) and an abductive proof-procedure is used to determine whether observed events satisfy these constraints. However, ICs in $\mathscr{S}$CIFF are only deterministic, with limited applicability in domains where event occurrence is uncertain. Meanwhile, IFF^{Prob}, a probabilistic extension of the IFF [7] proof-procedure that supports non-ground abduction with probabilistic integrity constraints (PICs) was introduced [6]. Since the real world is both *uncertain* and *dynamic*, in this extended abstract we extend the IFF^{Prob} proof-procedure with the characterizing features of $\mathscr{S}$CIFF, obtaining a new probabilistic ALP framework named $\mathscr{S}$CIFF^{Prob}, that can deal both with dynamic event streams and PICs. Both features of $\mathscr{S}$CIFF^{Prob} are crucial for the probabilistic compliance task in declarative Process Mining [5, 12, 13], a very recent line of research aiming to handle uncertainty in declarative process models. In fact, $\mathscr{S}$CIFF^{Prob} is able to perform run-time verification of compliance of incrementally changing histories with respect to process models composed of PICs.

2 Background

The $\mathscr{S}$CIFF language [3] supports the expression of events, expectations about events and relationships between them. An event is an atom with predicate symbol **H**, whose first argument is a ground term representing the happened event, and whose second argument is an integer representing the time at

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 1–4, doi:10.4204/EPTCS.439.1

© M. Vespa, E. Bellodi, M. Gavanelli & E. Lamma
This work is licensed under the
Creative Commons Attribution License.

which the event has happened. An expectation is an abducible atom with predicate symbol **E**, whose first argument is a term representing the event, and whose second argument is a variable or an integer indicating at what time the event is expected. Expectations can include variables. An *execution trace* is a set of ground happened events, and is denoted with the symbol $\mathscr{H}$. Integrity Constraints (ICs) are logical implications that relate expectations, events and restrictions. For example, the IC: $\mathbf{H}(action1(a),T_1) \rightarrow \mathbf{E}(action2(a),T_2) \wedge T_2 > T_1 \vee \mathbf{E}(action3(a),T_3) \wedge T_3 > T_1$ states that if event $action1(a)$ (with a generic argument a) occurred at time T_1, then either *action2* or *action3* is expected to happen after T_1.

A IFF [7] program is a triple $ALP = \langle \mathbf{KB}, \mathbf{IC}, \mathscr{A} \rangle$, where **KB** is a logic program, $\mathscr{A}$ is a set of predicates, called *abducibles*, that are not defined in **KB** and whose truth cannot be inferred but only hypothesized, and **IC** is a set of implications that must always be satisfied. The objective of abductive reasoning is to find a set $\Delta \subseteq \mathscr{A}$ of hypotheses that satisfies the goal G and the ICs: $KB \cup \Delta \models G$ and $KB \cup \Delta \models IC$. A $\mathscr{S}$CIFF^{Prob} program ALP^p is represented by the same triple above, except for the fact that the set **IC** is partitioned into a set $\mathbf{IC}^p$ of Probabilistic Integrity Constraints (PICs), and a set $\mathbf{IC}^{np}$ of non-probabilistic ICs. The syntax of a PIC is: $p_i :: Body \rightarrow Head$. According to the distribution semantics of Probabilistic Logic Programming [9], $\mathscr{S}$CIFF^{Prob} programs define a probability distribution over IFF programs called *worlds*, including each IC $\in \mathbf{IC}^{np}$ and some instances of any IC $\in \mathbf{IC}^p$ with probability p_i. When the body of the i_{th} PIC C_i is proven true, for a given substitution θ_j of the variables in the body, its consequences are not propagated in all possible explanations, but, instead, two alternative explanations are generated: one in which $C_i\theta_j$ is assumed to hold (the IC is added to the explanation with probability p_i, indicated by the triple $(C_i, \theta_j, 1)$) and its consequences are propagated, while in the other the PIC is not assumed to hold ($C_i\theta_j$ is removed from the explanation with probability $1 - p_i$) and the consequence is not derived.

3 $\mathscr{S}$CIFF^{Prob} Semantics

In order to enrich the IFF^{Prob} with the abilities to deal with a dynamic evolving execution trace and expected behavior, we define the *declarative* semantics, i.e., when a trace is compliant with a set of probabilistic specifications.

Definition 1 ($\mathscr{S}$CIFF^{Prob} Probabilistic Abductive Answer) *Given an ALP^p, the pair (Δ, θ) is a* probabilistic abductive answer *for a goal G in the explanation E and in the trace $\mathscr{H}$ if*

$$\begin{array}{ll} \boldsymbol{KB} \cup \mathscr{H} \cup \Delta \models G\theta & \forall (C_i, \theta_j, 1) \in E,\ \boldsymbol{KB} \cup \mathscr{H} \cup \Delta \models C_i\theta_j \\ \boldsymbol{KB} \cup \mathscr{H} \cup \Delta \models \boldsymbol{IC}^{np} & \boldsymbol{KB} \cup \mathscr{H} \cup \Delta \models \mathbf{E}(X,T) \rightarrow \mathbf{H}(X,T) \end{array} \quad \textit{In}$$

such a case, we write $ALP^p_{\mathscr{H}} \models^{\Delta}_{E} G$.

The *operational* semantics of $\mathscr{S}$CIFF^{Prob} is based on that of IFF^{Prob}, extended to support dynamically incoming events and expectations. We model incoming events by a queue $\mathscr{Q} = [\mathbf{H}(X_1,T_1), \mathbf{H}(X_2,T_2), \ldots]$. We indicate with $\mathscr{Q} = [\mathbf{H}(X,T) \mid \mathscr{Q}']$ the fact that the first event of $\mathscr{Q}$ is $\mathbf{H}(X,T)$ and that the remaining part of the queue is $\mathscr{Q}'$. $\mathscr{Q} = []$ is the empty queue.

When a new event arrives it is extracted from the queue and added to the current trace:

$$\frac{\mathscr{H} \quad [\mathbf{H}(X,T) \mid \mathscr{Q}']}{\{\mathbf{H}(X,T)\} \cup \mathscr{H} \quad \mathscr{Q}'} \quad (happening)$$

An expectation is fulfilled if it matches an incoming event:

$$\frac{\mathbf{H}(X,T_H) \quad \mathbf{E}(Y,T_E)}{X = Y,\ T_H = T_E,\ \mathbf{FULF}(\mathbf{E}(Y,T_E))} \quad \frac{\mathbf{H}(X,T_H) \quad \mathbf{E}(Y,T_E)}{X \neq Y \ \vee \ T_H \neq T_E} \quad (fulfillment)$$

If the queue is empty and an expectation $\mathbf{E}(X,T)$ remains unfulfilled, a violation occurs:

$$\frac{\mathbf{E}(X,T) \quad \mathbf{FULF}(\mathbf{E}(X,T)) \notin \Delta,\ \mathscr{Q} = []}{false} \quad (violation)$$

Definition 2 (Successful derivation) *A derivation node has the form $\langle R, \Delta, PSIC, \mathscr{Q}, \mathscr{H}, E, \theta \rangle$, where the goal still to be proven is partitioned into the set R of non-abducible literals and the set Δ of abducibles, PSIC is a set of (probabilistic and non probabilistic) ICs, $\mathscr{Q}$ is the dynamic events' queue, $\mathscr{H}$ the history, E the current explanation, and θ the current substitution. A derivation for goal G in an ALP^p succeeds if it starts from a node $\langle G, \emptyset, IC, \mathscr{Q}, \emptyset, \emptyset, \emptyset \rangle$ and terminates in a node $\langle \emptyset, \Delta, PSIC, [], \mathscr{H}, E, \theta \rangle$ with no transitions applicable. In this case $ALP^p_{\mathscr{Q} \mapsto \mathscr{H}}\ {}^{\mathscr{S}} \vdash^{\Delta}_{E} G\theta$.*

Theorem 1 (Soundness) *If there exists a successful derivation $ALP^p_{\mathscr{Q} \mapsto \mathscr{H}}\ {}^{\mathscr{S}} \vdash^{\Delta}_{E} G\theta$, then $ALP^p_{\mathscr{H}}\ {}^{\mathscr{S}} \models^{\Delta}_{E} G\theta$.*

Theorem 2 (Completeness) *If $ALP^p_{\mathscr{H}}\ {}^{\mathscr{S}} \models^{\Delta}_{E} G\theta$ then there exists a successful derivation $ALP^p_{\mathscr{Q} \mapsto \mathscr{H}}\ {}^{\mathscr{S}} \vdash^{\Delta'}_{E'} G\theta$, where $\Delta' \subseteq \Delta$, $\forall (C_i, \theta_j, 1) \in E'$, $(C_i, \theta_j, 1) \in E$ and $\mathscr{H} = \{\mathbf{H}(X,T) \mid \mathbf{H}(X,T) \in \mathscr{Q}\}$.*

4 An Application for Compliance Verification in Process Mining

Process Mining (PM) is defined as an analytical discipline for discovering, monitoring and improving business process models in organizations [2]. The data consist of execution traces of a process and *compliance* [1] focuses on establishing to what extent actual executions of a process are in line with the expected behavior of a reference model called *process model*. We consider here "declarative" process models, which specify the properties of a process through constraints. If compliance techniques are limited to static, a-posteriori analysis, the extent of (non-)conforming behavior is quantified after the process instance has been completed. In run-time compliance we quantify the (non-)conforming behavior as the execution is running (i.e., we have a stream of incoming activities). The $\mathscr{S}$CIFF^{Prob} proof-procedure

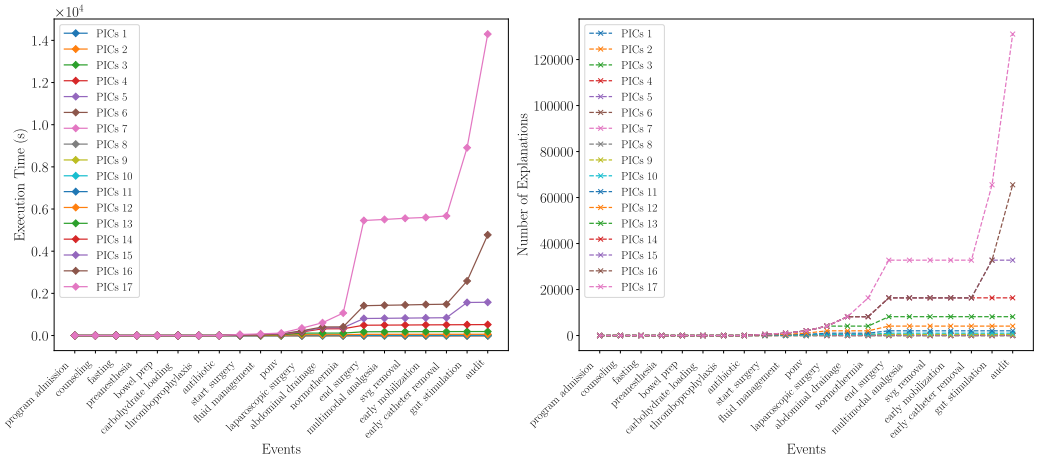


Figure 1: Execution time in s (left) and # of explanations (right) as the number of incoming events increases.

can be exploited in both cases to perform probabilistic verification of compliance. Our approach builds on previous formal results in probabilistic declarative PM [12, 13, 11]. We consider a process model inspired by a healthcare domain [10], made of 21 PICs capturing several constraints among events, from patient admission to post-surgery recovery.

A trace t represents a patient, composed of 21 temporally ordered activities. $\mathscr{S}$CIFF^{Prob} is capable of both performing a-posteriori compliance - returning the probability of compliance of t versus the model - and run-time compliance - returning the probability of compliance as events arrive incrementally and its value is updated after each new event. Evaluation of $\mathscr{S}$CIFF^{Prob} scalability in the run-time case is shown in Fig. 1.

References

[1] Wil M. P. van der Aalst (2011): *Process Mining: Discovery, Conformance and Enhancement of Business Processes*, 1st edition. Springer Publishing Company, Incorporated, doi:10.1007/978-3-642-19345-3.

[2] van der Aalst et al. (2011): *Process Mining Manifesto*. In Florian Daniel, Kamel Barkaoui & Schahram Dustdar, editors: *Business Process Management Workshops - BPM 2011 International Workshops, Clermont-Ferrand, France, August 29, 2011, Revised Selected Papers, Part I*, *Lecture Notes in Business Information Processing* 99, Springer, pp. 169–194, doi:10.1007/978-3-642-28108-2_19.

[3] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello & Paolo Torroni (2008): *Verifiable agent interaction in abductive logic programming: The SCIFF framework*. *ACM Trans. Comput. Log.* 9(4), pp. 29:1–29:43, doi:10.1145/1380572.1380578.

[4] Marco Alberti, Marco Gavanelli & Evelina Lamma (2013): *The CHR-based Implementation of the SCIFF Abductive System*. *Fundam. Informaticae* 124(4), pp. 365–381, doi:10.3233/FI-2013-839.

[5] Anti Alman, Fabrizio Maria Maggi, Marco Montali & Rafael Peñaloza (2022): *Probabilistic declarative process mining*. *Inf. Syst.* 109, p. 102033, doi:10.1016/J.IS.2022.102033.

[6] Elena Bellodi, Marco Gavanelli, Riccardo Zese, Evelina Lamma & Fabrizio Riguzzi (2021): *Nonground Abductive Logic Programming with Probabilistic Integrity Constraints*. *Theory and Practice of Logic Programming* 21(5), pp. 557–574, doi:10.1017/S1471068421000417.

[7] Tze Ho Fung & Robert A. Kowalski (1997): *The IFF Proof Procedure for Abductive Logic Programming*. *Journal of Logic Programming* 33(2), pp. 151–165, doi:10.1016/S0743-1066(97)00026-5.

[8] Antonis Kakas, Robert Kowalski & Francesca Toni (1992): *Abductive Logic Programming*. *J. Log. Comput.* 2, pp. 719–770, doi:10.1093/logcom/2.6.719.

[9] Taisuke Sato (1995): *A Statistical Learning Method for Logic Programs with Distribution Semantics*. In Leon Sterling, editor: *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995*, MIT Press, pp. 715–729.

[10] U. O. Gustafsson, et al. (2019): *Guidelines for Perioperative Care in Elective Colorectal Surgery: Enhanced Recovery After Surgery (ERAS®) Society Recommendations: 2018*. *World Journal of Surgery* 43(3), pp. 659–695, doi:10.1007/s00268-018-4844-y.

[11] Michela Vespa (2024): *A probabilistic semantics for process mining*. In Davide Bacciu & Ivan Donadello, editors: *Proceedings of the AIxIA Doctoral Consortium 2024 co-located with the 23rd International Conference of the Italian Association for Artificial Intelligence (AIxIA 2024)*, 3914, CEUR-WS, pp. 1–6.

[12] Michela Vespa, Elena Bellodi, Federico Chesani, Daniela Loreti, Paola Mello, Evelina Lamma & Anna Ciampolini (2024): *Probabilistic Compliance in Declarative Process Mining*. In: *Proceedings of the 3rd Int. Workshop on Process Management in the AI Era (PMAI 2024) @ECAI 2024, Santiago de Compostela, Spain, October 19, 2024*, *CEUR Workshop Proceedings* 3779, pp. 11–22. Available at `https://ceur-ws.org/Vol-3779/paper1.pdf`.

[13] Michela Vespa, Elena Bellodi, Federico Chesani, Daniela Loreti, Paola Mello, Evelina Lamma, Anna Ciampolini, Marco Gavanelli & Riccardo Zese (2024): *Probabilistic Traces in Declarative Process Mining*. In: *AIxIA 2024 - XXIIIrd Int. Conf. of the Italian Ass. for Artificial Intelligence, Proceedings*, *Lecture Notes in Computer Science* 15450, Springer, pp. 330–345, doi:10.1007/978-3-031-80607-0_25.

Propositional Abduction via Only-Knowing A Non-Monotonic Approach

Sanderson Molick
Division of Humanities
Federal Institute of Pará
Pará, Brazil
smolicks@gmail.com

Vaishak Belle
School of Informatics
University of Edinburgh*
Edinburgh, UK
vbelle@ed.ac.uk

The paper introduces a basic logic of knowledge and abduction by extending Levesque's logic of only-knowing with an abduction modal operator defined via the combination of basic epistemic concepts. The upshot is an alternative approach to abduction that employs a modal vocabulary and explores the relation between abductive reasoning and epistemic states of only-knowing. Furthermore, by incorporating a preferential relation into modal frames, we provide a non-monotonic extension of our basic framework capable of expressing different selection methods for abductive explanations. Core metatheoretic properties of non-monotonic consequence relations are explored within this setting and shown to provide a well-behaved foundation for abductive reasoning.

1 Introduction

Abductive reasoning is a central component for contemporary accounts of knowledge representation in artificial intelligence (see [22] or [21]). An abductive problem in a logic $\mathscr{L}$ is often represented by a background theory Θ along with an observed event α such that $\Theta \not\models_{\mathscr{L}} \alpha$. The solution for the problem described by the pair $\langle \Theta, \alpha \rangle$ is a formula φ (also called the *explanans* or the *explanation*) such that $\Theta \cup \{\varphi\} \models_{\mathscr{L}} \alpha$.

In this paper, we investigate the problem of selecting explanations φ that are coherent with an agent's background knowledge Θ, and that provide an explanatory account of an observed event α. Accordingly, we introduce a novel extension of Levesque's **Logic of Only-Knowing** ($\mathscr{OL}$) capable of expressing abductive reasoning within the epistemic bounds of the agent's background knowledge, a modal framework called the **Logic of Only-Knowing and Abduction** (herafter denoted as $\mathscr{AOL}$). In $\mathscr{AOL}$, abduction is treated as a derived modality defined through the interaction of basic epistemic concepts. The framework is constructed through the addition of an abduction operator **A** defined in terms of the epistemic modalities **O** and **K** (where $\mathbf{O}\varphi$ denotes that the agent "only-knows φ" and $\mathbf{K}\varphi$ denotes that the agent "knows φ"). As a result, abduction is understood as an epistemic process of knowledge acquisition, in accordance with Peircean accounts of abduction such as that explored by A. Aliseda ([1]).

The problem of developing knowledge-based accounts of abduction via $\mathscr{OL}$ was first introduced by Levesque ([11]) as a way to distinguish implicit from explicit beliefs. While other modal-based accounts of abduction, such as [17], address abduction within dynamic epistemic logics and related frameworks, our approach focuses on selecting explanations compatible with the agent's *only-known* background knowledge, thus modeling abduction as a structured form of inference grounded in basic epistemic states. To this end, the paper introduces the basic logic of only-knowing and abduction ($\mathscr{AOL}$) along with a

*Vaishak Belle was funded by a CISCO grant and a Royal Society University Research Fellowship.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 5–17, doi:10.4204/EPTCS.439.2

© Molick, S. & Belle, V.
This work is licensed under the
Creative Commons Attribution License.

non-monotonic extension (the logic $\mathscr{AOL}^{\neg}$) by enriching the underlying modal framework with a preferential structure. Furthermore, the formal properties of the abductive modality in terms of consistency, explanatory power, and minimality, are explored within our framework.

Overview. The syntax and semantics of our basic framework are presented in Section 2. Core properties of the abductive modality in capturing key aspects of logic-based accounts of abduction are also presented in Sections 2 and 3. Subsequently, in Section 4, we augment our modal frames via a plausibility semantics and introduce a non-monotonic consequence relation based on the logic of only-knowing and abduction. This move allows one to select minimal explanations compatible with a given abductive problem $\langle \Theta, \alpha \rangle$. The section also introduces a set-theoretic approach to minimal explanations that allows a different criterion for selecting minimal explanations. In Section 5, we prove core metatheoretic results for our non-monotonic framework. Finally, in Section 6, the paper ends with a discussion of related work and the paths for future exploration.

2 Preliminaries: the logic $\mathscr{AOL}$

Syntax

We consider a modal propositional language $\mathrm{L}_{\mathbf{O},\mathbf{A}}$ built from a countable set of atoms (At), the standard boolean connectives $\neg$, $\wedge$ and $\rightarrow$, and three epistemic operators: **K**, **O** and **A**. The set of formulas of $\mathrm{L}_{\mathbf{O},\mathbf{A}}$ (denoted by $Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$) is inductively constructed as follows:

$$\varphi :== p \,|\, \neg\varphi \,|\, \varphi \wedge \varphi \,|\, \varphi \rightarrow \varphi \,|\, \mathbf{K}\varphi \,|\, \mathbf{O}\varphi \,|\, \mathbf{A}\varphi,$$

where $p \in At$. We use ψ, φ, δ, ... to denote arbitrary individual formulas, and Γ, Δ, Θ, ... to denote sets of formulas in $Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$. We shall follow Levesque's presentation ([12]) and let $\mathbf{O}\varphi$ to be read as "the agent *only-knows* φ" and $\mathbf{K}\varphi$ to be read as "the agent *knows* or *believes* φ". In addition, $\mathbf{A}\varphi$ will be read as "the agent knows φ *by abduction*". A formula will be called **abductive** if it is a boolean formula preceded only by the modal operator **A**.

Semantics

We adopt Levesque's semantic convention, according to which an epistemic situation is represented by a pair $(\mathscr{W}, w)$, where $\mathscr{W}$ is a set of epistemic states and $w \in \mathscr{W}$ is designated as the **actual** state, with the usual truth-assignment to atomic formulas. The semantic structure is defined as follows:

Definition 2.1. *Where* $\mathrm{L}_{\mathbf{O},\mathbf{A}}$ *is a modal propositional language, a* ***Kripke model*** *for* $\mathrm{L}_{\mathbf{O},\mathbf{A}}$ *is a tuple* $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, v \rangle$*, where* $\mathscr{W}$ *is a set of epistemic states,* $\mathscr{R}$ *is a binary relation over* $\mathscr{W}$ *and* v *is a mapping that assigns to each atom of* $\mathrm{L}_{\mathbf{O},\mathbf{A}}$ *a subset of* $\mathscr{W}$*. The definition of* satisfaction *(or* truth*) of a formula in a state* $w \in \mathscr{W}$ *of a model* $\mathscr{M}$ *is determined by the following clauses:*

(i) $(\mathscr{M}, w) \models p$ *iff* $w \in v(p)$ *and* p *is an atom.*

(ii) $(\mathscr{M}, w) \models \neg\varphi$ *iff* $(\mathscr{M}, w) \not\models \varphi$.

(iii) $(\mathscr{M}, w) \models \varphi \wedge \psi$ *iff* $(\mathscr{M}, w) \models \varphi$ *and* $(\mathscr{M}, w) \models \psi$.

(iv) $(\mathscr{M}, w) \models \varphi \rightarrow \psi$ *iff* $(\mathscr{M}, w) \models \varphi$ *implies* $(\mathscr{M}, w) \models \psi$.

(v) $(\mathscr{M}, w) \models \mathbf{K}\varphi$ *iff* $(\mathscr{M}, w') \models \varphi$ *for all* $w' \in \mathscr{M}$ *such that* wRw'.

(vi) $(\mathscr{M}, w) \models \mathbf{O}\varphi$ *if for all states* $w' \in \mathscr{W}$*:* wRw' *iff* $(\mathscr{M}, w') \models \varphi$.

(vii) $(\mathscr{M},w) \models \mathbf{A}\varphi$ *iff* $\exists \alpha : (\mathscr{M},w) \models \mathbf{O}\alpha$ *and* $(\mathscr{M},w) \models \mathbf{K}(\varphi \rightarrow \alpha)$.

According to clause (vii), the abductive formula selects the explanation compatible with the agent's *only-known* background knowledge. The use of the formula α, combined with the modality **O**, ensures that the abductive inference relies solely on the agent's explicitly represented background knowledge, requiring no additional epistemic closure or stronger assumptions. The following example illustrates the rationale for the behavior of the modality **A**:

Example 1. *Let* Θ *denote a set of diagnostic principles represented as only-knowing implications, F denote the set of symptoms of a patient, and E be the set of possible diagnostic explanations. Consider the following medical situation*

$$\Theta = \{\mathbf{O}(cold \rightarrow cough),$$
$$\mathbf{O}(flu \rightarrow (cough \wedge fever)),$$
$$\mathbf{O}(pneumonia \rightarrow (chest_pain \wedge cough \wedge fever))\}$$
$$F = \{fever, cough\}$$
$$E = \{flu\}.$$

In this example, the best abductive explanation for the patient's symptom is the condition flu. *Let* $\mathscr{M}$ *be a model for the pair* $\langle \Theta, \alpha \rangle$ *and let* $\alpha := cough \wedge fever$. *Since both* $\mathbf{O}\alpha$ *and* $\mathbf{K}(\varphi \rightarrow \alpha)$ *hold, according to the semantic clause of the abduction operator* **A**, *the diagnostician can conclude the explanation* $\mathbf{A}\varphi$, *where* $\varphi = flu$. *In this framework, the doctor does not directly know that the patient has the condition* flu[1]. *Instead, the diagnosis arises as an* ***abductive inference****, derived from known symptoms and constrained hypotheses, given that the symptoms* $cough \wedge fever$ *are coherent with the hypothesis* flu. *The combination of* **K** *and* **O** *captures two distinct roles:* **O** *restricts the agent's knowledge base to certain hypotheses, while* **K** *provides the deductive link from hypotheses to observed symptoms (or events). The abductive conclusion* $\mathbf{A}\varphi$ *is a product of the agent's* only-known *knowledge base and what they can infer from it.*

The treatment of abduction as affirming the antecedent is a strategy pursued by different authors (such as [16], [1] or [8]) and central for modelling Peirce's notion of abduction. The kind of abduction illustrated by situations like Example 1 is also referred to by G. Schurz ([18]) as **selective abduction**, as it depends on selecting the best possible explanation from a predefined set of known hypotheses. According to the semantic clause of the **A** operator, the only-known particle selects the explanation that is part of the implicit knowledge of the agent.

As illustrated by the example, selecting explanations in accordance with the only-known background may prove useful in medical scenarios, where the explanation for a condition have to be selected without resorting to external sources of information. The use of formal methods for modelling abduction in medical scenarios is called **abductive diagnosis** by R. Brachman ([4]). The employment of the abductive modality **A** allows one to check for the best explanation compatible with the initial set of conditions and diagnostic principles. As a consequence, abductive diagnosis is treated as a form of selective abduction within our framework. We introduce additional definitions in the following:

Definition 2.2. *(Local consequence) We shall write* $\Gamma \models \varphi$ *to denote that for every Kripke model* $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, v \rangle$ *and every world* $w \in \mathscr{W}$, *if* $(\mathscr{M},w) \models \gamma$ *for all* $\gamma \in \Gamma$, *then* $(\mathscr{M},w) \models \varphi$.

Definition 2.3. *(Global validity) A formula* φ *will be called* ***globally valid*** *with respect to a class of models* $\mathscr{F}$ *(written* $\models^{\mathscr{F}} \varphi$*) if, for every model* $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, v \rangle$ *in* $\mathscr{F}$, *and for every world* $w \in \mathscr{W}$, $(\mathscr{M},w) \models \varphi$ *holds.*

[1] Note that even though *cold* could be abductively inferred, the condition flu is the most compatible with the set of symptoms.

Definition 2.4. *(Abductive explanation) A formula φ will be called an **explanation** for the abduction problem $\langle \Theta, \alpha \rangle$ if φ is an abductive formula and $\Theta \cup \{\varphi\} \models \alpha$.*

The following shows how to guarantee the existence of a formula α in case $(\mathscr{M}, w) \models \mathbf{A}\varphi$ holds. In what follows, given a kripke model $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, v \rangle$, let $R(w) = \{w' \in \mathscr{W} : wRw'\}$ and $R_\varphi(w) = \{w' \in R(w) : (\mathscr{M}, w') \models \varphi\}$.

Theorem 1. *Where Θ is a finite non-empty set of only-known formulas true in a model $\mathscr{M}$. The following statement holds for every $\phi \in Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$:*

- *If $(\mathscr{M}, w) \models \mathbf{A}\varphi$ holds, there is a formula α such that both $(\mathscr{M}, w) \models \mathbf{O}\alpha$ and $(\mathscr{M}, w) \models \mathbf{K}(\varphi \to \alpha)$ hold.*

Proof. Suppose $(\mathscr{M}, w) \models \mathbf{A}\varphi$ holds in a model $\mathscr{M}$. The proof consists in showing how α can be constructed for $\Theta \neq \emptyset$. Let $\Theta = \{\psi_1, \psi_2, ..., \psi_n\}$ be the set of true only-known formulas in a state w. By the semantic clause for only-knowing, we obtain that $(\mathscr{M}, w) \models \mathbf{O}\varphi$ iff $R_\varphi(w) \neq \emptyset$. Now let $\alpha := \bigwedge\{\varphi : (\mathscr{M}, w') \models \varphi$ and $w' \in R(w)\}$. Note that $\alpha = (\bigwedge \psi_i)$, for $1 \leq i \leq n$. From the construction, we know that $(\mathscr{M}, w') \models \varphi \Rightarrow (\mathscr{M}, w') \models \alpha$ holds in every w'. Hence $\varphi \to \alpha$ is true in all accessible states from w. As a result, by the semantic clauses for $\mathbf{K}$ and $\mathbf{O}\alpha$, $(\mathscr{M}, w) \models \mathbf{K}(\varphi \to \alpha)$ and $(\mathscr{M}, w) \models \mathbf{O}\alpha$ follow. □

The abduction operator is sensitive to the addition of new information in a model, a central feature for modeling abductive reasoning. Consider a model $\mathscr{M}_1$ with $\mathscr{W} = \{w_1\}$ and the following validities: $(\mathscr{M}, w_1) \models fever$, $(\mathscr{M}, w_1) \models flu$ and $(\mathscr{M}, w_1) \not\models cold$. Now let $\alpha := flu$ and $\varphi := fever$. Note that $(\mathscr{M}, w_1) \models \mathbf{O}flu$ and $(\mathscr{M}, w_1) \models fever \to flu$. Hence $(\mathscr{M}, w_1) \models \mathbf{K}(fever \to flu)$ and $(\mathscr{M}, w_1) \models \mathbf{A}fever$. Now consider $\mathscr{W} = \{w_1, w_2\}$ such that $w_1 R w_2$ and the following valuation: $(\mathscr{M}, w_2) \models fever$, $(\mathscr{M}, w_2) \not\models flu$ and $(\mathscr{M}, w_2) \models cold$. The conclusion $\mathbf{A}fever$ is blocked. The following are central properties for logic-based accounts of abduction (as proposed by Aliseda ([1])) and shown to be in accordance with the $\mathbf{A}$ modality:

Definition 2.5. *Let $\langle \Theta, \alpha \rangle$ be an abductive problem. The following states central properties for any $\langle \Theta, \alpha \rangle$:*
(Consistency) $\Theta \cup \varphi$ is consistent and $\Theta, \varphi \models \alpha$;
(Explainability) (i) $\Theta \not\models \alpha$; (ii) $\varphi \not\models \alpha$; and (iii) $\Theta, \varphi \models \alpha$.

Theorem 2. *Let $\langle \Theta, \alpha \rangle$ be an abductive problem, where Θ is a finite set of only-known implications. The following statements are true for any $\Theta \cup \alpha \subseteq Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$:*
(i) If $(\mathscr{M}, w) \models \mathbf{A}\varphi$ holds, then Explainability is true for $\langle \Theta, \alpha \rangle$
(ii) If $(\mathscr{M}, w) \models \mathbf{A}\varphi$ holds, then Consistency is true for $\langle \Theta, \alpha \rangle$.

Proof. Suppose $(\mathscr{M}, w) \models \mathbf{A}\varphi$ holds in a model $\mathscr{M}$. Theorem 1 and the semantic clause for $\mathbf{A}$ are sufficient for meeting the conditions for explainability.
For Consistency, assume for a contradiction that $\Theta \cup \varphi$ is inconsistent. Hence, there is a formula $\neg\varphi \in \Theta$ such that $(\mathscr{M}, w) \models \neg\varphi$. Given that Θ is a set of only-known implications, we know that φ has the form $\mathbf{O}(\alpha \to \beta)$, for some $\alpha, \beta \in Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$. Therefore $(\mathscr{M}, w) \models \neg\mathbf{O}(\alpha \to \beta)$, i.e., there is a state w' such that wRw' and $(\mathscr{M}, w') \not\models (\alpha \to \beta)$. Now, by the semantic clause for $\mathbf{A}$, we know that $(\mathscr{M}, w) \models \mathbf{K}(\alpha \to \beta)$. As a consequence, $(\mathscr{M}, w') \models (\alpha \to \beta)$ holds for every w' such that wRw', and a contradiction follows. □

3 Core properties of abduction

In this section, we introduce some core properties of $\mathscr{AOL}$. We start by noting that the modalities $\mathbf{K}$ and $\mathbf{A}$ do not collapse. While $\models \mathbf{O}\varphi \rightarrow \mathbf{A}\varphi$ holds, we have $\not\models \mathbf{A}\varphi \rightarrow \mathbf{O}\varphi$. We start by defining the set of models capable of validating instances of abductive inference.

Proposition 1. *For every $\varphi, \psi \in Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$: $\mathbf{O}\varphi, \mathbf{O}(\psi \rightarrow \varphi) \models \mathbf{A}\psi$ holds in all reflexive models.*

Proof. Let $\mathscr{M}$ be a reflexive model such that $\mathscr{W} = \{w_1, w_2\}$ and $\mathscr{R} = \{w_1 R w_2, w_1 R w_1, w_2 R w_2\}$. Now assume that $(\mathscr{M}, w) \models \mathbf{O}(\psi \rightarrow \varphi)$ and $(\mathscr{M}, w) \models \mathbf{O}\varphi$ hold. Note that since w is reflexive, it follows that $(\mathscr{M}, w) \models (\psi \rightarrow \varphi)$ and $(\mathscr{M}, w) \models \mathbf{K}(\psi \rightarrow \varphi)$. Let $\alpha := \varphi$. By the semantic clause for $\mathbf{A}$, we obtain $(\mathscr{M}, w) \models \mathbf{A}\psi$.

□

Proposition 2. *The following inferences are valid in the logic of only-knowing and abduction $\mathscr{AOL}$:*

(i). $\mathbf{A}\varphi, \mathbf{A}\psi \models \mathbf{A}(\varphi \wedge \psi)$
(ii). $\mathbf{A}\varphi, \mathbf{A}\psi \models \mathbf{A}(\varphi \vee \psi)$
(iii). $\mathbf{A}\varphi, \mathbf{A}(\varphi \rightarrow \delta) \models \mathbf{A}\delta$

Proof. Suppose $(\mathscr{M}, w) \models \mathbf{A}\varphi$ and $(\mathscr{M}, w) \models \mathbf{A}\psi$ hold in a given model $\mathscr{M}$. By the semantic clause for $\mathbf{A}$, there is an α such that $(\mathscr{M}, w) \models \mathbf{O}\alpha$ and $(\mathscr{M}, w) \models \mathbf{K}(\varphi \rightarrow \alpha)$. By the same reasoning, there is a β such that $(\mathscr{M}, w) \models \mathbf{O}\beta$ and $(\mathscr{M}, w) \models \mathbf{K}(\psi \rightarrow \beta)$. Now let $\alpha := \varphi$ and $\beta := \psi$. By the semantic clause for $\mathbf{O}$, $(\mathscr{M}, w) \models \varphi \wedge \psi$. Now set $\gamma = (\varphi \wedge \psi)$. Hence $(\mathscr{M}, w) \models \mathbf{O}\gamma$ and $(\mathscr{M}, w) \models \mathbf{K}(\gamma \rightarrow \gamma)$. Finally, the semantic clause for $\mathbf{A}$ allows us to conclude $(\mathscr{M}, w) \models \mathbf{A}(\varphi \wedge \psi)$. The proof of (ii) and (iii) follows by analogous reasoning. □

It is worth highlighting that the $\mathscr{OL}$-fragment is unable to account for interesting instances of abductive inferences based on only-known formulas in reflexive models. Based on the model $\mathscr{M}$ from Proposition 1, it is easy to see that $(\mathscr{M}, w) \models \mathbf{O}(\psi \rightarrow \varphi)$ and $(\mathscr{M}, w) \models \mathbf{O}\varphi$, but also $(\mathscr{M}, w) \not\models \mathbf{O}\psi$. Although some models without reflexivity may also validate Proposition 1, restricting $\mathscr{AOL}$ to reflexive model can be helpful to avoid problematic cases such as the following: let $\mathscr{M}$ be a model such that $\mathscr{W} = \{w_1, w_2\}$, $\mathscr{R} = \{w_1 R w_2, w_2 R w_1\}$ and the following valuation: $(\mathscr{M}, w_1) \models \alpha$, $(\mathscr{M}, w_1) \models \varphi \rightarrow \alpha$, $(\mathscr{M}, w_2) \not\models \alpha$ and $(\mathscr{M}, w_2) \not\models \varphi \rightarrow \alpha$. By the semantic clause for $\mathbf{A}$, we obtain $(\mathscr{M}, w_2) \models \mathbf{K}(\varphi \rightarrow \alpha)$ and $(\mathscr{M}, w_2) \models \mathbf{A}\varphi$. However, one may also have $(\mathscr{M}, w_2) \not\models \varphi$. In models like this, the agent is allowed to infer by abduction a formula that is false at some state. By augmenting the model via reflexivity, one cannot obtain both $(\mathscr{M}, w_2) \models \mathbf{K}(\varphi \rightarrow \alpha)$ and $(\mathscr{M}, w_2) \not\models \varphi \rightarrow \alpha$. Unfortunately, even **S5**-models of $\mathscr{AOL}$ are not sufficient for ensuring non-vacuity of abduction as follows:

Proposition 3 (Non-Vacuity of Abduction)**.** *For every $\varphi \in Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$: If $(\mathscr{M}, w) \models \mathbf{A}\varphi$, then $(\mathscr{M}, w) \not\models \mathbf{K}\neg\varphi$.*

One should note, however, that asserting $(\mathscr{M}, w) \models \mathbf{A}\varphi$ is not equivalent to $(\mathscr{M}, w) \models \varphi$. As a result, $(\mathscr{M}, w) \models \mathbf{A}\varphi$ and $(\mathscr{M}, w) \models \neg\varphi$ are not logically contradictory. Even in models where the conditional $\varphi \rightarrow \alpha$ holds vacuously true, the agent may still wish to consider what can be coherently inferred based on the basis of her only-known background knowledge alone. Situations like this may also arise in medical diagnosis, where the available information may allow the diagnostician to entertain mutually incompatible explanatory hypotheses. In the absence of definitive evidence, the diagnostician must reason abductively based on her only-known background knowledge, while remaining aware that

some of the entertained hypotheses are incompatible with some evidence or may ultimately prove to be false. In Section 4, we avoid models with vacuously true conditional by augmenting our modal framework with a preferential semantics. The following result shows that models where non-vacuity of abduction fails have an equivalent submodel where non-vacuity holds.

Lemma 1. *Let $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, v \rangle$ be a Kripke model and $w \in \mathscr{W}$ a state such that $(\mathscr{M}, w) \models \mathbf{A}\varphi$. Then there exists a submodel $\mathscr{M}' = \langle \mathscr{W}', \mathscr{R}', w' \rangle$, and $w' \in \mathscr{W}'$ for which the following holds:*

$$(\mathscr{M}', w') \models \mathbf{A}\varphi \text{ and } (\mathscr{M}', w') \not\models \mathbf{K}\neg\varphi \text{ hold iff } \mathscr{W}' = \{w' \in W \mid (\mathscr{M}, w') \models \Theta \text{ and } (\mathscr{M}, w') \not\models \neg\varphi\} \text{ is non-empty,}$$

where Θ denotes the agent's knowledge base.

Proof. Let $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, v \rangle$ be a model with a state w such that $(\mathscr{M}', w') \models \mathbf{A}\varphi$ and $(\mathscr{M}', w') \not\models \mathbf{K}\neg\varphi$ hold. Define the submodel model $\mathscr{M}' = \langle \mathscr{W}', \mathscr{R}', v' \rangle$ as follows:

- $\mathscr{W}' = \{w' \in W \mid (\mathscr{M}, w') \models \Theta \text{ and } (\mathscr{M}, w') \not\models \neg\varphi\}$,
- $\mathscr{R}' = \mathscr{R} \cap (\mathscr{W}' \times \mathscr{W}')$,
- $v' = v(p) \cap \mathscr{W}'$,

By construction, it is easy to see that $\mathscr{W}'$ is non-empty. From l.h.s. to r.h.s., from the fact that $(\mathscr{M}, w') \models \Theta$ holds and the assumption, $(\mathscr{M}', w') \models \mathbf{A}\varphi$ follows. □

Lemma 2. *For any $\varphi \in Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$, it holds that:*

$$\text{If } (\mathscr{M}, w) \models \varphi, \text{ then } (\mathscr{M}', w') \models \varphi,$$

where $\mathscr{M}'$ is the submodel of $\mathscr{M}$.

Proof. By induction on the length of φ. For $\varphi = p$, where p is an atom. By construction, $v'(p) = v(p) \cap \mathscr{W}'$. Thus, $w \in v(p)$ implies $w' \in v'(p)$. Therefore $(\mathscr{M}', w') \models p$. For $\varphi = \neg\psi$. By induction hypothesis, $(\mathscr{M}, w) \models \phi$ implies $(\mathscr{M}', w') \models \phi$. Hence, $(\mathscr{M}, w) \not\models \phi$ implies $(\mathscr{M}', w') \not\models \phi$. Given the clause for negation, $(\mathscr{M}', w) \models \neg\phi$. The rest of the proof follows as usual. The case of modal operators follows by the restriction of the accessibility relation.

□

Theorem 3. *Where $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, v \rangle$ is a Kripke model in which non-vacuity of abduction fails at a state $w \in \mathscr{W}$. There exists a submodel $\mathscr{M}' = \langle \mathscr{W}', \mathscr{R}', v' \rangle$ and $w' \in \mathscr{W}'$ such that the following holds:*
(i) $(\mathscr{M}', w') \models \Theta$, for any $\Theta \subseteq Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$ such that $(\mathscr{M}, w) \models \Theta$; and
(i) $(\mathscr{M}', w') \models \mathbf{A}\varphi$ implies $(\mathscr{M}', w') \not\models \mathbf{K}\neg\varphi$.

Proof. Assume $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, v \rangle$ is a model in which non-vacuity of abduction fails at a state $w \in \mathscr{W}$. By Lemma 1, there exists a submodel $\mathscr{M}' = \langle \mathscr{W}', \mathscr{R}', v' \rangle$. From Lemma 2, there is a state $w' \in \mathscr{W}'$ such that $(\mathscr{M}, w) \models \varphi$ implies $(\mathscr{M}', w') \models \varphi$. Moreover, $(\mathscr{M}', w') \models \Theta$. Now assume $(\mathscr{M}', w') \models \mathbf{A}\varphi$. By construction of $\mathscr{M}'$, $(\mathscr{M}', w') \not\models \mathbf{K}\neg\varphi$ follows. □

4 Abductive problems and minimal explanations

The task of selecting explanations that are compatible with an agent's background knowledge often requires comparing them according to specific criteria.[2] In particular, the ability to evaluate and compare alternative sets of explanations proves especially valuable in scenarios where multiple explanations are available for the same event (Aliseda in [2] points out how selection of minimal explanations is a central property of logic-based approaches to abduction). For example, as illustrated by Example 1, if the doctor discovers that the cough is not a revelant symptom, she must reconsider which explanation to adopt. Although the logic $\mathscr{AOL}$ is sensitive to new information at the level of individual models, the logic remains monotonic at the level of the consequence relation.

In this section, we augment our Kripke models with a plausibility relation $\prec$. The resulting logic is a non-monotonic extension of $\mathscr{AOL}$ (hereafter denoted as $\mathscr{AOL}^{\prec}$). Modeling abductive reasoning via non-monotonic consequence relations can be found in [16] or [15] via adaptive logics. While these authors treat abduction through non-monotonic ampliative approaches to validate inferences of the form $(\varphi \wedge \psi), \psi \models \varphi$, the $\mathscr{AOL}^{\prec}$ framework focuses on inferences compatible with the only-knowing background. We start by introducing basic definitions.

Definition 4.1. *Where* $\mathrm{L}_{\mathbf{O},\mathbf{A}}$ *is a modal language, a* ***plausibility Kripke model*** *for* $\mathrm{L}_{\mathbf{O},\mathbf{A}}$ *is a tuple* $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, \prec, v \rangle$*, where* $\mathscr{W}$ *is a set of epistemically possible states,* $\mathscr{R}$ *is a binary relation over* $\mathscr{W}$*,* $\prec$ *is a plausibility ordering over* $\mathscr{W}$*, where* $w \prec w'$ *means that* w *is* ***preferred (more plausible)*** *than* w'*, and* v *is a mapping that assigns to each atom of* $\mathrm{L}_{\mathbf{O},\mathbf{A}}$ *a subset of* $\mathscr{W}$*. The definition of* satisfaction *(or* truth*) of a formula in a state* $w \in \mathscr{W}$ *of a model* $\mathscr{M}$ *remains similar to that of Definition 2.1. The altered clauses are the following:*

(iv) $(\mathscr{M}, w) \models \varphi > \psi$ *iff* $(\mathscr{M}, w) \models \psi$ *holds in all* $\prec$*-minimal states accessible from* w *such that* $(\mathscr{M}, w) \models \varphi$*.*

(vii) $(\mathscr{M}, w) \models \mathbf{A}\varphi$ *iff* $\exists \alpha : (\mathscr{M}, w) \models \mathbf{O}\alpha$ *and* $(\mathscr{M}, w) \models (\varphi > \alpha)$ *in all* $\prec$*-minimal states accessible from* w*.*

Hereafter we shall assume that $\prec$ is a transitive and connected relation (i.e., $w \prec w'$ or $w' \prec w$ holds for all $w, w' \in \mathscr{W}$).

Definition 4.2. *A state* w *is called* ***$\prec$-minimal*** *(or just* ***minimal****) if there is no* $w' \prec w$ *and it is called* ***φ-minimal*** *in case* $(\mathscr{M}, w) \models \varphi$ *and there is no* $w' \in \mathscr{W}$ *such that* $w' \prec w$ *and* $(\mathscr{M}, w') \models \varphi$*.*

Definition 4.3. *A formula* φ *will be called a* ***preferential consequence*** *of a set of formulas* Γ *(denoted* $\Gamma \models_{\prec} \varphi$*) iff* $(\mathscr{M}, w) \models \varphi$ *holds in all minimal states* w *such that* $(\mathscr{M}, w) \models \gamma$ *for every* $\gamma \in \Gamma$*.*

Definition 4.4. *(Minimal explanation) A formula* φ *will be called a* ***minimal explanation*** *for the abduction problem* $\langle \Theta, \alpha \rangle$ *if* φ *is an abductive formula and* $\Theta \cup \{\varphi\} \models_{\prec} \alpha$*.*

It is easy to see that Consistency and Explainability hold for abductive problems in the preferential consequence $\models_{\prec}$. Moreover, the semantic clause for the preferential conditional guarantees non-vacuity of abduction. The following example illustrates a situation where the $\mathscr{AOL}$ consequence relation fails to select a single explanation, while the preferential entailment can do it:

Example 2. *Let* $\langle \Theta, \beta \rangle$ *be an abduction problem where* $\Theta = \{((\alpha > (\gamma \vee \delta)) \vee \delta) \vee (\psi > ((\gamma \vee \delta) \vee \gamma))\}$ *and* $\beta = (\alpha \vee \psi)$*. Consider a plausibility model* $\mathscr{M}$ *such that* $\mathscr{W} = \{w_1, w_2\}$*,* $\mathscr{R} = \{w_1 R w_1, w_2 R w_2\}$ *with* $w_1 \prec w_2$ *and the following scenario:*

[2] See, for instance, [15] and [6], who argue that selective abduction involves some form of defeasible reasoning.

$\mathscr{W}$	$(\alpha > (\gamma \vee \delta)$	$(\psi > (\gamma \vee \delta)$	γ	δ	$\mathbf{A}\alpha$	$\mathbf{A}\psi$
w_1	$\surd$	$\times$	$\surd$	$\times$	$\surd$	$\times$
w_2	$\times$	$\surd$	$\times$	$\surd$	$\times$	$\surd$

According to the above example, we have $(\mathscr{M}, w_1) \models \gamma$ *and therefore* $(\mathscr{M}, w_1) \models \gamma \vee \delta$*. Moreover, by the semantic clause for implication,* $(\mathscr{M}, w_1) \models \alpha > (\gamma \vee \delta)$*. By the semantic clause* $\mathbf{O}$*,* $(\mathscr{M}, w_1) \models \mathbf{O}(\gamma \vee \delta)$ *follows. Finally, the semantic clause for* $\mathbf{A}$ *gives us* $(\mathscr{M}, w_1) \models \mathbf{A}\alpha$*. By following the same rationale for the state* w_2*, we obtain* $(\mathscr{M}, w_2) \models \mathbf{A}\psi$*. Now, the* $\mathscr{AOL}$*-consequence relation is unable to select one explanation. However, since* w_1 *is a minimal state, the preferential consequence will select* $\mathbf{A}\alpha$ *as the minimal explanation.*

4.1 Non-monotonic properties

The properties of the order $\prec$ guarantees that the preferential consequence constructed over the logic $\mathscr{AOL}$ enjoy core metaproperties for non-monotonic logics (see [20] or [13]) and thus consists in a well-behaved consequence relation[3]. The proofs of each of the following theorems are standard and can be found in [13].

Theorem 4 (Supraclassicality)**.** *For any* $\Gamma \cup \{\varphi\} \subseteq Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$*: If* $\Gamma \models \varphi$ *then* $\Gamma \models_{\prec} \varphi$*.*

Theorem 5 (Reflexivity)**.** *For any* $\Gamma \cup \{\varphi\} \subseteq Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$*: If* $\varphi \in \Gamma$ *then* $\Gamma \models_{\prec} \varphi$*.*

Theorem 6 (Cautious Monotony)**.** *For any* $\Gamma \cup \{\varphi, \psi\} \subseteq Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$*: If* $\Gamma \models_{\prec} \psi$ *and* $\Gamma \models_{\prec} \varphi$*, then* $\Gamma \cup \{\varphi\} \models_{\prec} \psi$*.*

Theorem 7 (Cautious Transitivity)**.** *For any* $\Gamma \cup \{\varphi, \psi\} \subseteq Fm(\mathrm{L}_{\mathbf{O},\mathbf{A}})$*: If* $\Gamma \models_{\prec} \varphi$ *and* $\Gamma \cup \{\varphi\} \models_{\prec} \psi$*, then* $\Gamma \models_{\prec} \psi$

4.2 Additional ways of selecting explanations

In this section, we introduce additional restrictions for comparing different sets of minimal explanations relative to a given abduction problem. Example 2 illustrated a situation in which more than one explanation was available relative to the abduction problem. In these situations, the $\mathscr{AOL}$-consequence is unable to select a single explanation. Due to the fact that $w_1 \prec w_2$, the preferential consequence allows us to select a unique explanation. In scenarios where multiple explanations are available, the preferential consequence allows not only the selection of minimal explanations, but also the ability to impose additional constraints on the set of minimal explanations[4]. In the following, we show that different standards for the selection of minimal explanations can be expressed via the logic $\mathscr{AOL}^{\prec}$.

Definition 4.5. *A set of formulas* Δ *will be called a* ***set of minimal explanations*** *for the abduction problem* $\langle \Theta, \alpha \rangle$ *if every* $\delta \in \Delta$ *is a minimal explanation (also written* Δ_{min}*).*

In what follows, we introduce different ways of comparing minimal explanations by introducing further restrictions on the set of explanations selected by the preferential consequence $\models_{\prec}$. Let $\Pi(\Theta, \alpha) = \{\varphi \,|\, \Gamma, \varphi \models_{\prec} \alpha\}$, i.e., $\Pi(\Theta, \alpha)$ denotes the set of all minimal explanations of the problem $\langle \Theta, \alpha \rangle$ in a model $\mathscr{M}$.

[3] Here a non-monotonic logic $\mathscr{L}_{nm}$ is defined as a structure $\langle \models_m, \prec \rangle$, where $\models_m$ is a non-trivial monotonic consequence relation and $\prec$ is a preferential semantics.

[4] Different ways of comparing explanations via logic-based approaches to abduction are found in [7] via a probabilistic semantics or in [14] via a possible-world semantics.

Definition 4.6. *(Subset-minimality selection) Let* $\Pi(\Theta,\alpha)^{\subset} = \{\Delta \in \Pi(\Theta,\alpha) \mid \forall \Delta' \subset \Delta, \Delta' \notin \Pi(\Theta,\alpha)\}$. *The set* Δ *is called the* ***subset-minimal*** *set of explanations.*

Definition 4.7. *(Cardinality selection) Let* $\Pi(\Theta,\alpha)^{<} = \{\Delta \in \Pi(\Theta,\alpha) \mid \forall \Delta'$ *such that* $|\Delta'| < |\Delta|, \Delta' \notin \Pi(\Theta,\alpha)\}$. *The set* Δ *is called the* ***cardinality-minimal*** *set of explanations.*

For the priorization comparison we enhance the sets of minimal explanations via a priorization function $g : I \to \Delta$ such that $\delta_i \leq \delta_k$ or $\delta_k \leq \delta_i$ for each $i,k \in \Delta$ (and I is a countable index set). We will say that the index i is the **priority level** of the explanation δ_i. Given two sets of minimal explanations Δ_1 and Δ_2, we shall write $\Delta_1 \sqsubseteq \Delta_2$ in case $i \leq k$ for some $\delta_i \in \Delta_1$ and $\delta_k \in \Delta_2$.

Definition 4.8. *(Priorization selection) Let* $\Pi(\Theta,\alpha)^{\sqsubseteq} = \{\Delta \in \Pi(\Theta,\alpha) \mid \forall \Delta' \sqsubseteq \Delta, \Delta' \notin \Pi(\Theta,\alpha)\}$. *The set* Δ *is called the* ***priorization*** *set of explanations.*

The following example illustrates the differences between each standard of selection.

Example 3. *Consider the following medical situation:*

$$\begin{gathered}\Theta = \{\mathbf{O}(common_cold \to (sore_throat \vee cough)),\\ \mathbf{O}(strep_throat \to (sore_throat \wedge fever)),\\ \mathbf{O}(allergies \to (headache \vee itchy_eyes))\}\\ F = \{fever, sore_throat, headache\}\\ E = \{common_cold, sthrep_throat, allergies\}.\end{gathered}$$

Let $\Pi(\Theta,F) = \{\Delta_1,\Delta_2\}$, *where* $\Delta_1 = \{allergies, sthrep_throat\}$ *and* $\Delta_2 = \{common_cold, sthrep_throat\}$. *Both the cardinality and the subset selections are unable to pick a single set of minimal explanations. One way of handling this kind of situation is via the priorization selection. For this, assign the following priority levels:* $allergies_1$, $sthrep_throat_2$ *and* $common_cold_3$. *According to the priorization selection, the set* Δ_1 *will be selected as the most plausible for the abduction problem. The priorization approach may prove useful in medical scenarios where the treatment of a certain symptom needs to receive priority over others.*

By further restricting the set of explanations selected by the preferential consequence, one obtains the following additional consequence relations: $\models^{s}_{\prec}$, $\models^{c}_{\prec}$ and $\models^{p}_{\prec}$, where each consequence relation may be introduced as follows:

Definition 4.9. *(s-consequence)* $\Gamma \models^{s}_{\prec} \varphi$ *iff* $\Delta \subseteq \Gamma$, *where* $\Delta \in \Pi(\Theta,\varphi)^{\subset}$ *and* $\Delta \models_{\prec} \varphi$.

Definition 4.10. *(c-consequence)* $\Gamma \models^{c}_{\prec} \varphi$ *iff* $\Delta \subseteq \Gamma$, *where* $\Delta \in \Pi(\Theta,\varphi)^{<}$ *and* $\Delta \models_{\prec} \varphi$.

Definition 4.11. *(p-consequence)* $\Gamma \models^{p}_{\prec} \varphi$ *iff* $\Delta \subseteq \Gamma$, *where* $\Delta \in \Pi(\Theta,\varphi)^{\sqsubseteq}$ *and* $\Delta \models_{\prec} \varphi$.

In the next section, we show under which conditions each $\models^{*}_{\prec}$, where $* \in \{s,c,p\}$, coincides with $\models_{\prec}$. Furthermore, Theorem 7 in the next section guarantees that $\Pi(\Theta,\alpha) \neq \emptyset$ for any abduction problem $\langle\Theta,\alpha\rangle$. The table below shows the properties for each notion of consequence relation.

Example 4. *According to Example 4, while it follows that* $\Delta_1 \models allergies$ *and* $\Delta_1 \models sthrep_throat$, *it also follows* $\Delta_1, sthrep_throat \not\models allergies$. *Therefore, cautious monotony fails for the c- and s-consequence. To see that cautious transitivity fails, note that* $\Delta_1 \models allergies$ *and* $\Delta_1, allergies \models sthrep_throat$. *However, it also follows* $\Delta_1 \not\models allergies \to sthrep_throat$. *The remaining cases are proved by analogous reasoning.*

Property	$\models$	$\models_{\prec}$	$\models^{s}_{\prec}$	$\models^{c}_{\prec}$	$\models^{p}_{\prec}$
Supraclassicality	✓	✓	✓	✓	✓
Reflexivity	✓	✓	✓	✓	✓
Cautious Monotony	×	✓	×	×	×
Cautious Transitivity	×	✓	✓	✓	×

Table 1:

5 Metatheory

This section introduces central results for the metatheory of the non-monotonic $\mathscr{AOL}$, namely, we prove the existence of a minimal model for any abductive problem $\langle \Theta, \alpha \rangle$ within the framework of plausibility Kripke models, and show the conditions under which the selection method coincides with the preferential consequence relation. We begin by stating the following theorem:

Theorem 8 (Minimal Model Existence)**.** *For any abductive problem $\langle \Theta, \alpha \rangle$ with a plausibility Kripke model $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, \prec, v \rangle$, there exists a minimal model $\mathscr{M}_{min} = \langle \mathscr{W}_{min}, \mathscr{R}_{min}, v_{min} \rangle$, where $\mathscr{W}_{min}$ is the set of all α-minimal states in $\mathscr{W}$, and $(\mathscr{M}_{min}, w) \models \alpha$.*

Proof of minimal model existence

The proof consists in showing that given an arbitrary model $\mathscr{M}$ it is possible to construct a minimal model $\mathscr{M}_{min}$ that validates the same formulas. We start by introducing basic definitions.

Definition 5.1. *Given a plausibility Kripke model $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, \prec, v \rangle$, define the minimal model $\mathscr{M}_{min} = \langle \mathscr{W}_{min}, \mathscr{R}_{min}, v_{min} \rangle$ as follows:*

- $\mathscr{W}_{min} = \{ w \in \mathscr{W} \mid \forall w' \in \mathscr{W}, w' \prec w \implies w' = w \}$,
- $\mathscr{R}_{min} = \mathscr{R} \cap (\mathscr{W}_{min} \times \mathscr{W}_{min})$,
- $v_{min} = v(p) \cap \mathscr{W}_{min}$,

Lemma 3. *The model $\mathscr{M}_{min} = \langle \mathscr{W}_{min}, \mathscr{R}_{min}, \prec, v_{min} \rangle$ is minimal with respect to the preferential relation $\prec$.*

Proof. Straight from Definition 5.1 and construction of the model $\mathscr{M}_{min}$. □

Lemma 4. *For any state $w \in \mathscr{W}_{min}$, $(\mathscr{M}, w) \models \varphi$ implies $(\mathscr{M}_{min}, w) \models \varphi$.*

Proof. Let $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, \prec, v \rangle$ be a plausibility Kripke model, and let $\mathscr{M}_{\min} = \langle \mathscr{W}_{\min}, \mathscr{R}_{\min}, v_{\min} \rangle$ be the minimal model constructed as in Definition 5.1. The proof is done by induction on the length of φ. For case $\varphi = p$, where p is an atom. By construction, $v_{\min}(p) = v(p) \cap \mathscr{W}_{\min}$. Thus, $w \in v_{\min}(p)$ if and only if $w \in v(p)$. Therefore $(\mathscr{M}_{\min}, w) \models p$. For case $\varphi = \neg\psi$. By induction hypothesis, $(\mathscr{M}, w) \models \phi$ implies $(\mathscr{M}_{\min}, w) \models \phi$. Hence, $(\mathscr{M}, w) \not\models \phi$ implies $(\mathscr{M}_{\min}, w) \not\models \phi$. Given the clause for negation, $(\mathscr{M}_{\min}, w) \models \neg\phi$. For case $\varphi = K\psi$ or $O\psi$, recall that the accessibility relation $\mathscr{R}_{\min}$ is a restriction of $\mathscr{R}$ to $\mathscr{W}_{\min}$. Hence it follows that $(\mathscr{M}, w) \models \psi$ implies $(\mathscr{M}_{\min}, w) \models \psi$.

□

Theorem 9 (Minimal Model Existence)**.** *For any abductive problem $\langle \Theta, \phi \rangle$ with a plausibility Kripke model $\mathscr{M} = \langle \mathscr{W}, \mathscr{R}, \prec, v \rangle$, there exists a minimal model $\mathscr{M}_{min} = \langle \mathscr{W}_{min}, \mathscr{R}_{min}, v_{min} \rangle$, where $\mathscr{W}_{min}$ is the set of all ϕ-minimal states in $\mathscr{W}$, and $(\mathscr{M}_{min}, w) \models \phi$.*

Proof. Assume $\langle \Theta, \varphi \rangle$ is an abduction problem in a model $\mathscr{M}$. By Lemma 4, we know there is minimal model $\mathscr{M}_{min}$ that validates $\langle \Theta, \varphi \rangle$. Now assume there is a solution α to the abduction problem $\langle \Theta, \varphi \rangle$ such that $(\mathscr{M}, w) \models \alpha$. Hence $(\mathscr{M}_{min}, w) \models \alpha$. By definition of minimality, α is a minimal explanation. □

Selection of minimal sets and preferential consequence

The following results show the conditions under which some selection method for minimal sets of explanations coincides with the preferential consequence relation relative to an abductive problem $\langle \Theta, \alpha \rangle$.

Definition 5.2. *Given an abduction problem $\langle \Theta, \alpha \rangle$, a set of explanations $\Delta \subseteq \Pi(\Theta, \alpha)$ is **subset-minimal** if there is no proper subset $\Delta' \subset \Delta$ such that $\Theta \cup \Delta' \models \alpha$.*

Lemma 5. *Let Δ be a subset-minimal set of explanations for the abduction problem $\langle \Theta, \alpha \rangle$. Therefore $\Theta \cup \Delta \models_{\prec} \alpha$.*

Proof. Let Δ be a subset-minimal set of explanations and assume $(\mathscr{M}, w) \models \Theta, (\mathscr{M}, w) \models \Delta$, and $(\mathscr{M}, w) \models \alpha$ hold for a state $w \in \mathscr{W}$. Since Δ is subset minimal, for any proper subset $\Delta' \subset \Delta$, $\Theta \cup \Delta' \not\models \alpha$. As a consequence, w is α-minimal under $\prec$, for there is no $w' \prec w$ such that $(M, w') \models \Theta$ and $(M, w') \models \alpha$. Finally, by definition of preferential consequence $\Theta \cup \Delta \models_{\prec} \alpha$ follows. □

Lemma 6. *Let $\Theta \cup \Delta \models_{\prec} \alpha$. Then Δ is subset-minimal.*

Proof. Assume $\Theta \cup \Delta \models_{\preceq} \alpha$ holds and let Δ be the set of explanations corresponding to a α-minimal state in which the following are valid: $(\mathscr{M}, w) \models \Theta, (\mathscr{M}, w) \models \Delta$, and $(\mathscr{M}, w) \models \alpha$. Now, assume for a contradiction that Δ is not subset-minimal. Hence there is a proper subset $\Delta' \subset \Delta$ such that $\Theta \cup \Delta' \models \alpha$. Moreover, there is a state $w' \in \mathscr{W}$ such that $w' \prec w$ and validates the following: $(\mathscr{M}, w') \models \Theta, (\mathscr{M}, w') \models \Delta$, and $(\mathscr{M}, w') \models \alpha$. However, by the connectedness of $\prec$ and the definition of minimality, a contradiction follows. □

Theorem 10. *Given a set Δ of minimal explanations for the abduction problem $\langle \Theta, \alpha \rangle$, Δ is subset-minimal iff $\Theta \cup \Delta \models_{\preceq} \alpha$.*

Proof. By Lemmas 3 and 4. □

Corollary 1. *For any abductive problem $\langle \Theta, \alpha \rangle$, if the plausibility relation $\prec$ on the set of states $\mathscr{W}$ is transitive and connected, the preferential consequence relation $\models_{\preceq}$ coincides with subset minimality in selecting minimal explanations.*

Analogous results can be proved for cardinality and the priorization selections of sets of minimal explanations. In fact, one can show that the cardinality and the subset-minimality selections always coincide in selecting minimal explanations for a finite knowledge base.

6 Related work and future direction

The logics of only-knowing were introduced by H. Levesque in [12] and later explored by distinct authors ([10], [5] and [3]) as a way of modelling different aspects of knowledge dynamics. In particular, the problem of developing knowledge-based accounts of abduction via $\mathscr{OL}$ was first introduced by Levesque in [11]. In this paper, we explored an alternative account by treating abduction as an epistemic

modality defined via the combination of basic epistemic concepts and within the bounds of the background knowledge of the agent. Furthermore, a non-monotonic consequence relation was built over the monotonic logic $\mathscr{AOL}$ by augmenting the semantic framework via plausibility Kripke models.

Different modal approaches to abduction are found in authors such as [14], [12] and [9]. While most current approaches to abduction seek to develop its dynamic reasoning through a process of discovery modeled by proof-theoretical means (see [14] or [19]), our work introduced a semantical approach via the introduction of a new modality in a modal (and preferential) structure. The development of adequate proof-theoretic tools for the resulting logic is a topic for future exploration. Furthermore, while a first connection between $\mathscr{OL}$ and non-monotonic formalism was introduced by Levesque ([11]) through an interpretation of autoepistemic logic, we introduced an alternative interpretation via $\mathscr{AOL}$ within a preferential modal semantics. Our framework represents a novel approach compared to other non-monotonic modal frameworks for abductive reasoning (such as [16]).

The results demonstrated in Section 4 and 5 show that the non-monotonic logic built over the $\mathscr{AOL}$ is capable of expressing different methods for selection of minimal explanations. The fact that $\mathscr{AOL}$ and $\mathscr{AOL}^{\prec}$ enjoy central properties for logic-based characterization of abduction suggests that these systems can be provided with sequent rules for abduction (in the style proposed by [2]). Furthermore, other expansions of $\mathscr{OL}$ to multi-modal framework are introduced by Levesque in [11]. The generalization of our framework to multi-agent scenarios or to other multi-modal frameworks for only-knowing remains a path for future exploration.

7 Acknowledgements

The authors thank the careful reading of the reviewers.

References

[1] Atocha Aliseda (2000): *Abduction as Epistemic Change: A Peircean Model in Artificial Intelligence*, pp. 45–58, doi:10.1007/978-94-017-0606-3_3. Springer Netherlands, Dordrecht.

[2] Atocha Aliseda (2006): *Abductive reasoning*. *Synthese Library* 330, Springer.

[3] Vaishak Belle & Gerhard Lakemeyer (2010): *Multi-agent only-knowing revisited*. In: *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*, pp. 49–59. Available at https://cdn.aaai.org/ocs/1361/1361-7395-1-PB.pdf.

[4] Ronald J Brachman (2004): *Knowledge Representation and Reasoning*. Morgan Kaufman/Elsevier.

[5] Jianhua Chen (1994): *The logic of only knowing as a unified framework for non-monotonic reasoning*. *Fundamenta Informaticae* 21(3), pp. 205–220.

[6] Claudio Delrieux (2004): *Abductive inference in defeasible reasoning: a model for research programmes*. *Journal of Applied Logic* 2(4), pp. 409–437, doi:10.1016/j.jal.2004.07.003. CMSRA.

[7] Thomas Eiter & Georg Gottlob (1995): *The complexity of logic-based abduction*. *Journal of the ACM (JACM)* 42(1), pp. 3–42, doi:10.1145/200836.200838.

[8] Peter A Flach & Antonis C Kakas (2000): *Abduction and Induction: Essays on their relation and integration*. *Applied Logic Series* 18, Springer Science & Business Media.

[9] Tjerk Gauderis (2013): *Modelling abduction in science by means of a modal adaptive logic*. *Foundations of Science* 18(4), pp. 611–624, doi:10.1007/s10699-012-9293-8.

[10] Joseph Y Halpern & Gerhard Lakemeyer (2001): *Multi-agent Only Knowing*. *Journal of Logic and Computation* 11(1), pp. 41–70. Available at https://arxiv.org/pdf/cs.AI/0001015.

[11] Hector J Levesque (1989): *A Knowledge-Level Account of Abduction.* In: *IJCAI*, pp. 1061–1067. Available at https://www.ijcai.org/Proceedings/89-2/Papers/034.pdf.

[12] Hector J. Levesque (1990): *All I know: A study in autoepistemic logic*. *Artificial Intelligence* 42(2), pp. 263–309, doi:10.1016/0004-3702(90)90056-6.

[13] David Makinson (2003): *Bridges between classical and nonmonotonic logic*. *Logic Journal of IGPL* 11(1), pp. 69–96.

[14] Marta Cialdea Mayer & Fiora Pirri (1995): *Propositional abduction in modal logic*. *Logic Journal of the IGPL* 3(6), pp. 907–919. Available at https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=fcbe4de867d95512288c2070fd594cf1d03ff090.

[15] Joke Meheus (2007): *Adaptive logics for abduction and the explication of explanation-seeking processes*. In: *International meeting on Abduction and the Process of Scientific Discovery*, Centro de Filosofia das Ciências da Universidade de Lisboa, pp. 97–119.

[16] Joke Meheus, Liza Verhoeven, Maarten Van Dyck & Dagmar Provijn (2002): *Ampliative adaptive logics and the foundation of logic-based approaches to abduction*. *Logical and computational aspects of model-based reasoning*, pp. 39–71.

[17] Angel Nepomuceno-Fernández, Fernando Soler-Toscano & Fernando R Velázquez-Quesada (2017): *Abductive reasoning in dynamic epistemic logic*. *Springer handbook of model-based science*, pp. 269–293.

[18] Gerhard Schurz (2008): *Patterns of abduction*. *Synthese* 164, pp. 201–234, doi:10.1007/s11229-007-9223-4.

[19] Fernando Soler-Toscano, Ángel Nepomuceno-Fernández & Atocha Aliseda-Llera (2009): *Abduction via C-tableaux and δ-resolution*. *Journal of Applied Non-Classical Logics* 19(2), pp. 211–225, doi:10.3166/jancl.19.211-225.

[20] Christian Strasser & G. Aldo Antonelli (2024): *Non-monotonic Logic*. In Edward N. Zalta & Uri Nodelman, editors: *The Stanford Encyclopedia of Philosophy*, Winter 2024 edition, Metaphysics Research Lab, Stanford University.

[21] Richmond Thomason (2024): *Logic-Based Artificial Intelligence*. In Edward N. Zalta & Uri Nodelman, editors: *The Stanford Encyclopedia of Philosophy*, Spring 2024 edition, Metaphysics Research Lab, Stanford University.

[22] Efthymia Tsamoura, Timothy Hospedales & Loizos Michael (2021): *Neural-symbolic integration: A compositional perspective*. In: *Proceedings of the AAAI conference on artificial intelligence*, 35, pp. 5051–5060, doi:10.1609/aaai.v35i6.16639.

An ASP-Based Framework for MUSes

Mohimenul Kabir
National University of Singapore

Kuldeep S Meel
Georgia Institute of Technology

Given an unsatisfiable formula, understanding the core reason for unsatisfiability is crucial in several applications. One effective way to capture this is through the minimal unsatisfiable subset (MUS), the subset-minimal set of clauses that remains unsatisfiable. Current research broadly focuses on two directions: (i) enumerating as many MUSes as possible within a given time limit, and (ii) counting the total number of MUSes for a given unsatisfiable formula.

In this paper, we introduce an answer set programming-based framework, named MUS-ASP, designed for *online* enumeration of MUSes. ASP is a powerful tool for its strengths in knowledge representation and is particularly suitable for specifying complex combinatorial problems. By translating MUS enumeration into answer set solving, MUS-ASP leverages the computational efficiency of state-of-the-art ASP systems. Our extensive experimental evaluation demonstrates the effectiveness of MUS-ASP and highlights the acceleration in both MUS enumeration and counting tasks, particularly when integrated within hybrid solvers, including the framework proposed in this paper. The extended version of this paper is available at: `https://arxiv.org/abs/2507.03929`.

1 Introduction

When dealing with an unsatisfiable formula — one that has no solution — it is often crucial to identify the reasons for its unsatisfiability. A common concept in the literature is the *minimal unsatisfiable subset* (MUS), which provides a concise explanation for the formula's unsatisfiability [64]. For instance, in *requirement analysis* [16], a set of constraints represents the requirements that need to be fulfilled. If these constraints are unsatisfiable, a MUS highlights the minimal set of requirements that cannot be satisfied simultaneously.

Given an unsatisfiable Boolean formula F, expressed as a conjunction of clauses $\{C_1, \ldots, C_{\mathsf{ncl}}\}$, a minimal unsatisfiable subset (MUS) of F is a subset $F' \subseteq F$ such that F' is unsatisfiable and for every $C_i \in F'$, the set $F' \setminus \{C_i\}$ is satisfiable. Finding MUSes is an active research area due to their wide range of applications, including formal equivalence checking [29], Boolean function bi-decomposition [26], counter-example guided abstraction refinement [6], circuit error diagnosis [43], and others [44, 47, 48].

Since the MUS of an unsatisfiable formula is not necessarily unique, it is important to enumerate multiple MUSes. Enumerating more MUSes of F provides a clearer explanation of the underlying unsatisfiability [19]. Thus, existing research focuses on enumerating MUSes in an *online fashion* [8, 19, 20, 57, 62, 78], aiming to identify as many MUSes as possible within a given time limit. Moreover, the MUS count is frequently used as a metric to quantify inconsistency in diagnostic tasks [44, 75].

Existing MUS enumeration algorithms can be broadly divided into two categories: algorithms based on the well-known *seed-shrink* scheme [18] and those employing the *hitting set duality* [81]. The seed-shrink scheme systematically explores subsets of F and identifies an unsatisfiable subset called a *u-seed*, which is then shrunk into a MUS through a single MUS *extractor* [14]. Conversely, the hitting set duality leverages the well-known relationship between MUSes and MCSes, where a *minimal correction subset* (MCS) is the minimal subset $F' \subset F$ such that the formula $F \setminus F'$ is satisfiable. This approach involves computing and exploiting the MCSes of F to enumerate MUSes [65]. Traditional MUS enumeration

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 18–33, doi:10.4204/EPTCS.439.3

© Kabir and Meel
This work is licensed under the
Creative Commons Attribution License.

techniques often rely on SAT-based methods. In contrast, we explore an alternative perspective by formulating MUS enumeration as an *answer set solving* problem, leveraging the expressiveness of ASP for modeling combinatorial structures.

Answer set programming (ASP) [67] is a well-known declarative problem-solving paradigm in knowledge representation and reasoning, valued for its ability to model complex combinatorial problems [35]. An ASP *program* consists of *logical rules* defined over *propositional atoms*, encoding domain knowledge and queries. An assignment of values to propositional atoms that satisfies the ASP semantics of the program is referred to as an *answer set*. ASP programs can be classified into two categories based on their expressiveness: *disjunctive* and *normal* ASP programs [61]. However, the expressiveness comes at the cost of computational complexity. Specifically, determining whether a disjunctive program has an answer set is $\sum_2^P$-complete [34], while the complexity reduces to NP-complete for normal programs [68].

The main contribution of the paper is the proposal of an ASP solving-based MUS enumeration framework called MUS-ASP. At a high level, given an unsatisfiable Boolean formula F, we formulate a disjunctive ASP program $\mathcal{P}_F$ such that the answer sets of $\mathcal{P}_F$ correspond to the unsatisfiable cores of F. To compute MUSes, we focus on *subset minimal* answer sets with respect to a *target* set of atoms. Computing the subset minimal answer sets is not a new research problem [3, 23, 39]; it has been shown to be useful in addressing various real-world computational problems [5, 32, 79]. We are not the first to leverage ASP for MUS computation; it has previously been used to compute MUSes of LTL_f [45]. Particularly, we harness the efficiency of subset minimal answer set solvers to compute MUSes. Moreover, we incorporate several heuristics into our encoding to enhance practical scalability. Although the worst-case complexity of disjunctive ASP is high ($\sum_2^P$-complete), practical instances often admit a *small backdoor* to *normality* [36], meaning that they can be transformed into normal programs by few atoms. From the computational complexity perspective, finding a MUS is in FP^{NP} [27, 70]. It is worth noting that reducing one problem to a harder one is not uncommon in combinatorial problem-solving [60]. For example, the MUS counting framework AMUSIC [21] invoked a $\sum_3^P$ oracle to find MUSes.

Building on prior work in ASP-based counting [54, 56], we provide an extensive experimental evaluation demonstrating that MUS-ASP outperforms existing MUS enumerators and counters across multiple benchmarks, where counting is performed via enumeration. Although MUS-ASP does not consistently dominate across every benchmark, our proposed *hybrid* approach, which integrates MUS-ASP with existing enumerators, achieves significant performance gains across all benchmarks. While the *complete* MUS enumeration is theoretically intractable [83], MUS-ASP significantly outperforms state-of-the-art MUS counters in practice.

The paper is organized as follows: Section 2 presents the necessary background knowledge, Section 3 reviews the existing literature on MUS enumeration and counting, Section 4 presents our proposed framework MUS-ASP, designed for MUS enumeration, Section 5 presents the experimental results, finally, we conclude the paper in Section 6 with some future research directions.

2 Preliminaries

In this section, we present the background knowledge from propositional logic, hitting set duality, and answer set programming to understand the technical contribution.

Propositional Logic A *propositional variable* v takes a value from the domain $\mathbb{B} = \{0,1\}$, equivalently $\{\mathsf{false}, \mathsf{true}\}$. A *literal* ℓ is either a variable v (positive literal) or its negation $\neg v$ (negative literal). A *clause* C is a *disjunction* of literals, denoted as $C = \bigvee_i \ell_i$. A Boolean formula F, in *Conjunctive Normal*

Form (CNF), is a *conjunction* of clauses, represented as $F = \bigwedge_j C_j$. We use the notation $\mathsf{Var}(F)$ to denote the set of propositional atoms or variables within F. An assignment τ is a mapping $\tau : X \to \mathbb{B}$, where $X \subseteq \mathsf{Var}(F)$. For a variable $v \in X$, we define $\tau(\neg v) = 1 - \tau(v)$. An assignment τ satisfies a literal ℓ, denoted as $\tau \models \ell$, if τ evaluates ℓ to true. Similarly, an assignment τ satisfies a clause C, denoted as $\tau \models C$, if τ evaluates one of its literals to true. The assignment τ over $\mathsf{Var}(F)$ is a *model* of F if τ evaluates F to be true. A formula F is said to be *satisfiable* if there exists a model τ of F. Otherwise, the formula F is said to be *unsatisfiable*.

We consider a model of F as a set of positive literals it assigns and define the *minimal model* of a propositional formula F [7]. We also consider an *ordering operator* over two models [49]. For two given models τ_1 and τ_2, τ_1 is considered *smaller* than τ_2, denoted as $\tau_1 \leq \tau_2$, if and only if for each $x \in \mathsf{Var}(F)$, $\tau_1(x) \leq \tau_2(x)$; and τ_1 is *strictly smaller* than τ_2, denoted as $\tau_1 < \tau_2$, if $\tau_1 \leq \tau_2$ and $\tau_1 \neq \tau_2$. A model τ is a *minimal model* of F if and only if τ is a model of F and no model of F is strictly smaller than τ.

It is well known that in every minimal model τ, each variable assigned to true is *justified* [55]. More formally, given that τ is a minimal model of F, a (positive) literal $\ell \in \tau$ is said to be *justified* in τ, if there exists a clause $C \in F$ such that $\ell \in C$ and $\tau \setminus \{\ell\} \not\models C$. That is, every literal $\ell \in \tau$ must be justified; otherwise, if no such C exists for a particular ℓ, then $\tau \setminus \{\ell\}$ is also a model of F and $\tau \setminus \{\ell\} < \tau$, contradicting that τ is minimal. Intuitively, justification explains *why* each literal $\ell \in \tau$ must be in the model; removing it would violate the satisfaction of some clauses in F.

We consider a CNF formula as a multiset of clauses and use the operations introduced in the context of sets. Within the context of unsatisfiable Boolean formula, we introduce the following terminologies. An unsatisfiable core of F is a set of clauses $U \subseteq F$ such that U is unsatisfiable. A *minimal correction subset* (MCS) of F is a subset of clauses $F' \subset F$ such that $F \setminus F'$ is satisfiable and for every clause $C_i \in F'$, it holds that $F \setminus (F' \setminus \{C_i\})$ is unsatisfiable [64]. A *minimal unsatisfiable subset* (MUS) of F is a subset of clauses $F' \subseteq F$ such that F' is unsatisfiable and for every $C_i \in F'$, it holds that $F' \setminus \{C_i\}$ is satisfiable.

Example 1. *Consider the Boolean formula* $F = \{C_1 = \{x_1\}, C_2 = \{\neg x_1\}, C_3 = \{x_2\}, C_4 = \{\neg x_1, \neg x_2\}\}$. *The formula F is unsatisfiable. The formula F has* 5 *unsatisfiable cores:* $\{C_1, C_2\}$, $\{C_1, C_2, C_3\}$, $\{C_1, C_2, C_4\}$, $\{C_1, C_3, C_4\}$, $\{C_1, C_2, C_3, C_4\}$. *The formula F has* 3 *MCSes:* $\{C_1\}$, $\{C_2, C_3\}$, $\{C_2, C_4\}$. *The formula F has* 2 *MUSes:* $\{C_1, C_2\}$ *and* $\{C_1, C_3, C_4\}$.

Minimal Hitting Set Duality Given a collection of sets $\mathfrak{C}$, a hitting set H is a set such that for every $\mathfrak{C}' \in \mathfrak{C}$, $\mathfrak{C}' \cap H \neq \emptyset$. A hitting set is called minimal if none of its proper subsets is a hitting set. For a given Boolean formula F, the *minimal hitting set duality* [63, 81] says that H is a MUS of F if and only if H is a minimal hitting set of all MCSes of F.

Answer Set Programming An *answer set program P* consists of a set of rules, each rule r is structured as either of the two following forms:

$$r\colon a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, \mathsf{not}\ c_1, \ldots, \mathsf{not}\ c_n \tag{1}$$

$$r\colon \{a_1; \ldots; a_k\} \geq \mathsf{lb}(r) \leftarrow b_1, \ldots, b_m, \mathsf{not}\ c_1, \ldots, \mathsf{not}\ c_n \tag{2}$$

where, $a_1, \ldots, a_k, b_1, \ldots, b_m, c_1, \ldots, c_n$ are propositional variables or atoms; $k, m, n, \mathsf{lb}(r)$ are non-negative integers. The notations P and $\mathsf{atoms}(P)$ denote the rules and atoms within the program P, respectively. In rule r, the operator "not" denotes *default negation* [28]. For each rule r (shown in Equations (1) and (2)), we adopt the following notations: the atom set $\{a_1, \ldots, a_k\}$ constitutes the *head* of r, denoted

by $\mathsf{Head}(r)$, the set $\{b_1,\ldots,b_m\}$ is referred to as the *positive body atoms* of r, denoted by $\mathsf{Body}(r)^+$, and the set $\{c_1,\ldots,c_n\}$ is referred to as the *negative body atoms* of r, denoted by $\mathsf{Body}(r)^-$. A rule r is called a *constraint* when $\mathsf{Head}(r)$ has no atom. The *empty head* and *empty body* are often denoted by the notations $\bot$ (false) and $\top$ (true), respectively. A rule r is called a *cardinality* rule when the rule r is of form Equation (2). A program P is called a *disjunctive logic program* if there is a rule $r \in P$ such that r is of form eq. (1) and $|\mathsf{Head}(r)| \geq 2$ [15]. Otherwise, the program P is called a normal logic program. In this paper, we use the ASP-Core-2 standard input language for ASP programs [24].

In ASP, an interpretation τ over $\mathsf{atoms}(P)$ specifies which atoms are assigned true; an atom a is true under τ if and only if $a \in \tau$ (or false when $a \notin \tau$ resp.). An interpretation τ satisfies a rule r, denoted by $\tau \models r$, if and only if: (i) $(\mathsf{Head}(r) \cup \mathsf{Body}(r)^-) \cap \tau \neq \emptyset$ or $\mathsf{Body}(r)^+ \setminus \tau \neq \emptyset$ for a rule of the form eq. (1); and (ii) $|\mathsf{Head}(r) \cap \tau| \geq \mathsf{lb}(r)$ or $\tau \cap \mathsf{Body}(r)^- \neq \emptyset$ or $\mathsf{Body}(r)^+ \setminus \tau \neq \emptyset$ for a rule of the form eq. (2). An interpretation τ is a *model* of P, denoted by $\tau \models P$, if $\tau \models r$ for each $r \in P$. The *reduct* of a rule $r \in P$, with respect to an interpretation τ, denoted r^τ, is defined as (i) the singleton $\{\mathsf{Head}(r) \leftarrow \mathsf{Body}(r)^+ \mid \mathsf{Body}(r)^- \cap \tau = \emptyset\}$ if r is of the form eq. (1) and (ii) the set $\{a \leftarrow \mathsf{Body}(r)^+ \mid a \in \mathsf{Head}(r) \cap \tau, \mathsf{Body}(r)^- \cap \tau = \emptyset\}$ if r is of the form eq. (2). The *Gelfond-Lifschitz (GL) reduct* of a program P with respect to an interpretation τ, is defined as $P^\tau = \{r^\tau \mid r \in P\}$ [41]. The interpretation M is an *answer set* of P if $M \models P$ and no $M' \subset M$ exists such that $M' \models P^M$. We denote the answer sets of program P using the notation $\mathsf{AS}(P)$. We define the *subset minimal answer set* of a program P with respect to a set of atoms $O \subseteq \mathsf{atoms}(P)$. An answer set $M \in \mathsf{AS}(P)$ is a subset minimal answer set of P if there is no other answer set $M' \in \mathsf{AS}(P)$ such that $M' \cap O \subset M \cap O$ [3].

3 Related Works

Previous research on MUSes primarily focused on the extraction of a single MUS or computing one single MUS [12, 14, 76, 77]. However, with the increasing demand for computing multiple MUSes, the area of MUS enumeration has emerged as a significant focus of recent research [19]. Very recently, there has been growing interest in MUS counting.

MUS Enumeration. FLINT [78] utilizes a framework similar to those in [71] for MUS enumeration, iteratively computing MUSes and MCSes of an unsatisfiable formula across multiple *rounds*, alternatively *relaxing* an unsatisfiable formula and *strengthening* a satisfiable one.

MARCO [64], TOME [17], and ReMUS [20] are algorithms based on *seed-shrink* techniques. To find a MUS of a Boolean formula F, these algorithms gradually explore subsets of F to identify an unexplored subset that is unsatisfiable, called a *u-seed*. This u-seed is then shrunk to a MUS through a single MUS extraction subroutine. The main difference between these algorithms lies in how they identify the u-seed S. MARCO iteratively identifies maximal unexplored subsets of F and checks their satisfiability until a u-seed is found. TOME computes a chain of unexplored subsets of F — namely, $F_1 \subset \ldots F_p$, and then performs a *binary search* over the chain to identify MUSes. ReMUS employs a different technique to identify smaller u-seeds, which are easier to shrink. In particular, ReMUS finds the initial u-seed S among the maximal unexplored subsets of F and shrinks it into a MUS S'. To find subsequent MUSes, ReMUS selects a set R such that $S' \subset R \subset S$ and recursively searches for u-seeds within the maximal unexplored subsets of R, decreasing the size of the u-seed with each recursive call.

The hitting set duality is specifically used in MUS enumerators such as CAMUS [65] and DAA [9]. CAMUS operates in two phases: in the first phase, it computes MCSes of a formula using an *incremental* SAT solver; in the second phase, it enumerates MUSes by finding minimal hitting sets of the computed

MCSes through a *recursive branching algorithm*. DAA is based on the *Dualize and Advance* technique [42] and, unlike CAMUS, incrementally computes both MCSes and MUSes without enumerating all MCSes. A major drawback of these techniques is that the number of MCSes can grow exponentially.

The hitting set duality is also leveraged in the MUS enumerator MCS-MUS [8], which targets *constrained MCSes* instead of arbitrary ones. This approach incorporates several optimization techniques, such as *recursive model rotation* [12] and *clause set refinement* [77]. In a follow-up work, Bacchus and Katsirelos [57] extended MCS-MUS with a version called MCS-MUS-BT, which performs a backtracking search over a tree, exploiting information gathered from previous MUS enumerations to accelerate the finding of new MUSes.

MUS Counting. MUS counting without explicit enumeration was first introduced by [21]. They developed a *hashing-based* model counting framework, AMUSIC, which provides an (ε, δ)-approximation [25]. The counting framework AMUSIC does not scale due to its reliance on $\sum_3^P$ oracles. Later, Bendik and Meel [22] proposed CountMUST, a framework for exact MUS counting that leverages *subtractive reduction* [33] and *projected model counting*.

Subset-minimal Answer Sets. A variety of techniques exist for computing subset-minimal answer sets. The tool asprin [23] supports subset minimality as a special case of user-defined *preferences*. Clingo computes the subset-minimal answer sets using an approach similar to the OPTSAT algorithm [31]. More recently, Alviano et al. [3] proposed a family of methods for computing subset-minimal answer sets within the ASP solver Wasp.

4 Methodology

In this section, we present the MUS-ASP framework for MUS computation. We first introduce an ASP program $\mathcal{P}_F$ that captures the unsatisfiable cores of a given Boolean formula F, with each answer set corresponding to one such unsatisfiable core (Section 4.1). Then, we show the correctness of our methodology (Section 4.2). Finally, we describe some heuristics that improve performance by reducing the search space while preserving completeness (Section 4.3).

4.1 ASP Encoding of MUS-ASP

Since MUSes correspond to subset-minimal unsatisfiable subsets, and ASP excels at expressing subset minimality constraints, we encode the MUS enumeration problem as an ASP program. For a given Boolean formula F, the ASP program $\mathcal{P}_F$ defines the following atoms:

- λ_i: for each clause $C_i \in F$, we introduce a *clause selector atom* λ_i. Each interpretation τ over the atoms of $\mathcal{P}_F$ *selects* a subset of clauses from F; a clause C_i is selected by an interpretation τ if and only if $\lambda_i \in \tau$.
- p_x, n_x: for each variable $x \in \mathsf{Var}(F)$, we introduce two atoms: p_x and n_x.

  These atoms follow the symbolic interpretations: p_x (resp. n_x) indicates that the variable x is assigned true (resp. false). While a variable x cannot be assigned both values simultaneously, it is possible that for a given interpretation τ, both p_x and n_x belong to τ.
- unsat: we introduce an atom unsat to indicate that at least one of the clauses selected by an interpretation is not satisfied according to propositional semantics.

1 $\mathsf{p}_{x_i} \vee \mathsf{n}_{x_i} \leftarrow \top.$
2 $\mathsf{unsat} \leftarrow \lambda_i, \mathsf{n}_{x_1}, \ldots \mathsf{n}_{x_k}, \mathsf{p}_{x_{k+1}}, \ldots \mathsf{p}_{x_{k+m}}.$
3 $\bot \leftarrow$ not $\mathsf{unsat}.$
4 $\mathsf{p}_{x_i} \leftarrow \mathsf{unsat}.$
5 $\mathsf{n}_{x_i} \leftarrow \mathsf{unsat}.$
6 $\{\lambda_1; \ldots \lambda_{\mathsf{ncl}}\} \geq 0 \leftarrow \top.$

Encoding 1: Program $\mathcal{P}_F$.

We denote the set $\Lambda = \{\lambda_1, \ldots, \lambda_{\mathsf{ncl}}\}$ and the *projection* of τ onto the atom set Λ as $\tau_{\downarrow\Lambda}$. This projection extracts the selected clauses, allowing us to map each answer set to a corresponding unsatisfiable core. Given $F = \bigwedge_{i=1}^{\mathsf{ncl}} C_i$, we introduce the ASP program $\mathcal{P}_F$, where each unsatisfiable core of F corresponds one-to-one with the answer sets of $\mathcal{P}_F$. The ASP program $\mathcal{P}_F$ is outlined in Encoding 1.

The core idea behind $\mathcal{P}_F$ is to non-deterministically select a set of clauses from F, as specified by the rule in line 6. Among the selected clauses, the program enforces that at least one clause is unsatisfied under any variable assignment, ensuring the subset forms an unsatisfiable core (enforced by the rule in line 3). We formally prove that each answer set of $\mathcal{P}_F$ corresponds bijectively to an unsatisfiable core of F (see Theorem 1). More specifically, for every answer set $M \in \mathsf{AS}(\mathcal{P}_F)$, the projection $M_{\downarrow\Lambda}$ identifies an unsatisfiable core of F. For instance, if $M_{\downarrow\Lambda} = \{\lambda_1, \ldots, \lambda_k\}$, then the corresponding clause set $\{C_1, \ldots, C_k\} \subseteq F$ is unsatisfiable. Consequently, all subset minimal answer sets of $\mathcal{P}_F$ with respect to the set Λ capture exactly the minimal unsatisfiable subsets (MUSes) of F.

Theorem 1. *There is a bijective correspondence between the answer sets of $\mathcal{P}_F$ and the unsatisfiable cores of F, under the projection onto Λ.*

The proof of Theorem 1 is deferred to Section 4.2.

Proposition 1. *The subset minimal answer sets of $\mathcal{P}_F$ with respect to the atom set Λ correspond to the MUSes of F.*

Example 1 (continued)**.** *Given the Boolean formula $F = \{C_1 = \{x_1\}, C_2 = \{\neg x_1\}, C_3 = \{x_2\}, C_4 = \{\neg x_1, \neg x_2\}\}$, the program $\mathcal{P}_F$ consists of the following rules:*

$\mathsf{p}_{x_1} \vee \mathsf{n}_{x_1} \leftarrow \top.$ $\quad \mathsf{p}_{x_2} \vee \mathsf{n}_{x_2} \leftarrow \top.$
$\mathsf{unsat} \leftarrow \lambda_1, \mathsf{n}_{x_1}.$ $\quad \mathsf{unsat} \leftarrow \lambda_2, \mathsf{p}_{x_1}.$ $\quad \mathsf{unsat} \leftarrow \lambda_3, \mathsf{n}_{x_2}.$ $\quad \mathsf{unsat} \leftarrow \lambda_4, \mathsf{p}_{x_1}, \mathsf{p}_{x_2}.$
$\bot \leftarrow$ not $\mathsf{unsat}.$
$\mathsf{p}_{x_1} \leftarrow \mathsf{unsat}.$ $\quad \mathsf{n}_{x_1} \leftarrow \mathsf{unsat}.$ $\quad \mathsf{p}_{x_2} \leftarrow \mathsf{unsat}.$ $\quad \mathsf{n}_{x_2} \leftarrow \mathsf{unsat}.$
$\{\lambda_1; \lambda_2; \lambda_3; \lambda_4\} \geq 0 \leftarrow \top.$

The program $\mathcal{P}_F$ has 5 answer sets, each corresponding to an unsatisfiable core of F. Among them, two are subset minimal answer sets w.r.t. the atom set Λ : $\{\mathsf{p}_{x_1}, \mathsf{n}_{x_1}, \mathsf{p}_{x_2}, \mathsf{n}_{x_2}, \mathsf{unsat}, \lambda_1, \lambda_2\}$ and $\{\mathsf{p}_{x_1}, \mathsf{n}_{x_1}, \mathsf{p}_{x_2}, \mathsf{n}_{x_2}, \mathsf{unsat}, \lambda_1, \lambda_3, \lambda_4\}$. These two subset minimal answer sets correspond to the MUSes $\{C_1, C_2\}$ and $\{C_1, C_3, C_4\}$, respectively.

4.2 Theoretical Correctness

We outline the reduct of program $\mathcal{P}_F$ with respect to an arbitrary interpretation $\tau \models \mathcal{P}_F$ in Encoding 2. In both encodings, the lines starting with the % sign indicate comments. The reduct of $\mathcal{P}_F$ w.r.t. τ is important for the theoretical correctness of our technique (proof of Theorem 1). Following the approach

of SAT-based ASP solvers [61], the reduct replaces each rule $r \in \mathcal{P}_F$ by an *implication*: $\bigwedge_{b\in\mathsf{Body}(r)^+} b \wedge \bigwedge_{c\in\mathsf{Body}(r)^-} \neg c \longrightarrow \bigvee_{a\in\mathsf{Head}(r)} a$. These implications are written as disjunctions of literals in Encoding 2. For each interpretation $\tau \models \mathcal{P}_F$, we can show that $\{\mathsf{unsat}\} \subset \tau$ (Proposition 2). As a result, following the definition of reduct, there is no implication corresponding to the rule in line 3 (Encoding 1).

Proposition 2. *For each $\tau \models \mathcal{P}_F$: (i) $\{\mathsf{unsat}\} \subset \tau$ and (ii) $\forall x \in \mathsf{Var}(F), \{\mathsf{p}_x, \mathsf{n}_x\} \subset \tau$.*

Proposition 2 can be proved using *Clark completion* semantics [28]. If (i) is not true, then τ does not satisfy the rule in line 3 of Encoding 1. If (ii) is not true, then τ does not satisfy rules in lines 4 and 5.

Theorem 1. *There is a bijective correspondence between the answer sets of $\mathcal{P}_F$ and the unsatisfiable cores of F, under the projection onto Λ.*

Proof. The proof is structured into two parts:

1. For each $M \in \mathsf{AS}(\mathcal{P}_F)$, there exists a corresponding unsatisfiable core of F
2. For each unsatisfiable core U of F, there exists a corresponding answer set of $\mathcal{P}_F$

Proof of "Part 1": Given $M \in \mathsf{AS}(\mathcal{P}_F)$, without loss of generality, assume that $M_{\downarrow\Lambda} = \{\lambda_1, \ldots, \lambda_k\}$. We prove that the clause set $\{C_1, \ldots, C_k\}$ is an unsatisfiable core of F. Each answer set follows Clark completion semantics and according to Proposition 2, the answer set is $M = \{\lambda_1, \ldots, \lambda_k\} \cup \{\mathsf{unsat}\} \cup \{\mathsf{p}_x, \mathsf{n}_x | \forall x \in \mathsf{Var}(F)\}$. Note that for each answer set M, the set $M_{\downarrow\Lambda}$ maps to a unique set of clauses of F.

For the purpose of contradiction, assume that the clause set $\{C_1, \ldots, C_k\}$ is satisfiable. Let the clause set $\{C_1, \ldots, C_k\}$ have the variable set $\{x_1, \ldots, x_n\}$. Since the clause set $\{C_1, \ldots, C_k\}$ is satisfiable, there exists a model, which assigns variables $x_1, \ldots, x_{n^\star}$ to true and $x_{n^\star+1}, \ldots, x_n$ to false, where $n^\star \in [1, n]$. Now consider an interpretation $\tau' = \{\lambda_1, \ldots, \lambda_k, \mathsf{p}_{x_1}, \ldots, \mathsf{p}_{x_{n^\star}}, \mathsf{n}_{x_{n^\star+1}}, \ldots, \mathsf{n}_{x_n}\}$, which is a proper subset of M. There is a model of the clause set $\{C_1, \ldots, C_k\}$, meaning that the model satisfies each clause and we can easily show τ' satisfies $\mathcal{P}_F^M$, where $\mathcal{P}_F^M$ denotes the reduct of $\mathcal{P}_F$ with respect to M. Since $\tau' \subset M$, this contradicts that M is an answer set of $\mathcal{P}_F$.

Proof of "Part 2": Without loss of generality, let assume that the set of clauses $U = \{C_1, \ldots C_k\}$ is an unsatisfiable core of F. We prove that the unsatisfiable core U uniquely corresponds to an answer set of $\mathcal{P}_F$. More specifically, we prove that given the unsatisfiable core U, the interpretation $\tau = \{\lambda_1, \ldots, \lambda_k\} \cup \{\mathsf{unsat}\} \cup \{\mathsf{p}_x, \mathsf{n}_x | \forall x \in \mathsf{Var}(F)\}$ is an answer set of $\mathcal{P}_F$. Note that the interpretation τ is unique for each unsatisfiable core U. By definition, $\tau \models \mathcal{P}_F$. We prove that no $\tau' \subset \tau$ exists such that $\tau' \models \mathcal{P}_F^\tau$.

For the sake of contradiction, assume that $\exists \tau' \subset \tau$ such that τ' is a minimal interpretation satisfying $\mathcal{P}_F^\tau$. Recall from justification definition, as explained above, each atom in the minimal interpretation τ' is justified. Otherwise, there is another interpretation $\tau'' \subset \tau'$ that satisfies $\mathcal{P}_F^\tau$.

Note that $\{\lambda_1, \ldots, \lambda_k\} \subset \tau'$ (due to the clause introduced in line 5 of Encoding 2). Furthermore, $\mathsf{unsat} \notin \tau'$; if $\mathsf{unsat} \in \tau'$ (or unsat is justified), then $\tau' = \tau$ (due to the clauses in lines 3,4 of Encoding 2). For each variable $x \in \mathsf{Var}(F)$, it holds that $\{\mathsf{p}_x, \mathsf{n}_x\} \cap \tau' \neq \emptyset$ (due to the clause in line 1 of Encoding 2). Since τ' is a minimal interpretation satisfying $\mathcal{P}_F^\tau$, $\forall x \in \mathsf{Var}(F)$, we can say $\{\mathsf{p}_x, \mathsf{n}_x\} \not\subset \tau'$; because only one atom from $\{\mathsf{p}_x, \mathsf{n}_x\}$ can be justified from the clause in line 1 of Encoding 2. No further clause exists in Encoding 2 to justify both p_x and n_x. From the interpretation $\tau' \models \mathcal{P}_F^\tau$, we can derive an assignment over variables that satisfies each clause of U; otherwise, the atom unsat is justified in τ' due to clause in line 2 of Encoding 2 and it follows that $\tau = \tau'$ (it contradicts our assumption). The existence of a satisfying assignment of the clause set U contradicts that U is an unsatisfiable core. □

```
1    p_{x_i} ∨ n_{x_i}
2    ¬λ_i ∨ ¬n_{x_1} ∨ ... ∨ ¬n_{x_k} ∨ ¬p_{x_{k+1}} ∨ ... ∨ ¬p_{x_{k+m}} ∨ unsat
3    ¬unsat ∨ p_{x_i}
4    ¬unsat ∨ n_{x_i}
5    λ_i
```

Encoding 2: Reduct of $\mathcal{P}_F$ with respect to any interpretation $\tau \models \mathcal{P}_F$. The lines with % are comments.

4.3 Optimizing MUS Computation with Heuristics

Following the encoding $\mathcal{P}_F$, we introduce additional constraints to accelerate MUS computation. These optimization techniques are based on observations derived from MUS counting [22]. We incorporate these observations into the ASP encoding of $\mathcal{P}_F$ to enhance the efficiency of the MUS-ASP framework.

H_1 Union of MUSes. If we compute a union of MUSes or an over-approximation of them, then we can focus only on those clauses for MUS computation [66]. Computing the union of MUSes is computationally harder [72]. Building on the work in [22], we exploit a *lean kernel* based approach, by invoking a MaxSAT solver [58], to compute the over-approximation of all MUSes [69].

H_2 Minimum and Maximum Cardinalities of MUSes. If we compute the minimum (or a lower bound) and maximum (or an upper bound) cardinality of MUSes, then we can encode the cardinality information as an ASP cardinality rule [24] (Line 6 of Encoding 1) to reduce the search space of $\mathcal{P}_F$. Since finding the minimum cardinality MUS is intractable [46], we adopt the technique from [22] to compute a lower bound on the minimum MUS cardinality using the well-known minimal hitting set duality. Additionally, we compute an upper bound on the maximum MUS cardinality by solving a *maximum satisfiability* problem.

H_3 Component Decompositions. Bendik and Meel [22] partition the clauses into *disjoint components*, exploiting the fact that all clauses in a MUS originate from the same component. The computation of disjoint components involves constructing an undirected graph as follows: each clause C_i of the given formula corresponds to a node v_i and an edge exists between two nodes v_j and v_k, if there is a literal ℓ such that $\ell \in C_j$ and $\neg\ell \in C_k$. To reduce the search space based on the concept of disjoint components, MUS-ASP introduces constraints of the form: $\bot \leftarrow \lambda_i, \lambda_j$, where C_i and C_j are two clauses from different components in the graph constructed above. The constraint ensures that both atoms λ_i and λ_j cannot appear in the same interpretation.

H_4 Hitting Set Duality. Each MUS of F must intersect every known MCS. This observation is also utilized by [22]. Following a similar approach, we compute MCSes of F given a specific time limit. For each MCS represented by the clause set $\{C_{i_1}, \ldots, C_{i_k}\}$, we add the constraint: $\bot \leftarrow \mathsf{not}\ \lambda_{i_1}, \ldots, \mathsf{not}\ \lambda_{i_k}$, ensuring that at least one clause from each known MCS appears in every answer set.

H_5 Negative Literal Cover. If a clause C_i is in a MUS F' such that $\exists \ell \in C_i$, then there exists another clause $C_j \in F'$ such that $\neg\ell \in C_j$. We incorporate this observation into MUS-ASP as follows: let C_i be a clause in F such that $\ell \in C_i$ and let $C_{j_1}, \ldots, C_{j_k}$ be the clauses containing the literal $\neg\ell$, we introduce

a cardinality rule as: $\{\lambda_{j_1}; \ldots \lambda_{j_k}\} \geq 1 \leftarrow \lambda_i$, ensuring that at least one of the clauses containing $\neg\ell$ is selected when C_i is selected.

These heuristics constrain the search space of $\mathcal{P}_F$ to exclude some unsatisfiable cores of F that are not MUSes of F. Since these heuristics introduce some additional constraints to $\mathcal{P}_F$ that prune non-MUSes without excluding any MUS, thereby preserving the completeness of MUS enumeration [53]. Together, the encoding and heuristics enable MUS-ASP to efficiently enumerate MUSes by pruning the search space while ensuring completeness.

5 Experimental Evaluation

We evaluated MUS-ASP through extensive experiments on available MUS benchmarks. Our performance evaluation focused on addressing the following research questions:

RQ1 How does MUS-ASP compare to existing MUS enumerators?

RQ2 Can MUS-ASP outperform dedicated MUS counters in exact MUS counting?

RQ3 How does a hybrid integration of MUS-ASP improve efficiency across benchmarks?

Design Choice We prototyped MUS-ASP with Clingo [40], as the underlying ASP solver[1]. In experiment, we marked the Λ atoms (see Section 4.1) for minimization and invoked Clingo with the command `-enum-mode=domRec -heuristic=domain` to ensure subset minimality over Λ atoms. Note that we have also experimented with Wasp [4], which does not outperform Clingo on our benchmark set.

5.1 Benchmark, Baseline and Environmental Setup

To analyze **RQ1**, we collected our benchmarks from three sources: (i) the MUS track of the SAT 2011 competition[2] (ii) random graph coloring instances [66], and (iii) randomly generated formulas following the procedure outlined in [82], with settings consistent with [66]. We evaluated MUS-ASP against three off-the-shelf MUS enumeration tools: (i) MARCO [64] (ii) UNIMUS [19] and (iii) ReMUS [20].

To analyze **RQ2**, we considered the MUS counting benchmarks[1] [21, 22]. Clingo counts MUSes by enumeration. We baselined MUS-ASP against the exact MUS counter CountMUST [22]. We invoked CountMUST with the default parameter settings, which performed best in prior MUS counting evaluations [22]. We excluded AMUSIC from our comparison because it provides only approximate counts and was significantly outperformed by other MUS counters in previous studies. A direct comparison with existing ASP counters [52, 50] or minimal model counters [51] is not feasible, as MUSes correspond to subset-minimal answer sets with respect to a specific set of atoms.

We conducted our experiments on machines with AMD EPYC 7713 processors. Each benchmark instance was run on 1 core, 16GB memory, and a timeout of 3600 seconds.

Evaluation Metric. For **RQ1**, we evaluated the performance of these MUS enumerators based on the number of MUSes enumerated and their *average ranks*. We assign each solver a *rank* based on the number of MUSes it enumerates per instance (the lower ranks are better), as in [19]. For **RQ2**, we evaluated the MUS counters based on the number of instances solved and their PAR2 scores [10].

[1] The source code, dataset, and binaries are available at: `https://zenodo.org/records/15300472`

[2] `http://www.cril.univ-artois.fr/SAT11/`

5.2 Analysis of RQ1.

To address **RQ1**, we evaluated MUS-ASP with other MUS enumerators using their average ranks. The statistics are presented in Table 1, which indicate that MUS-ASP outperforms the existing MUS enumerators in terms of average ranks.

We compared the number of MUSes enumerated by MUS-ASP against other baseline tools (the comparison is visually presented in the extended version[3]). Our experimental analysis demonstrates that MUS-ASP enumerates more MUSes than MARCO, ReMUS, and UNIMUS in 555, 494, and 508 instances, respectively. Moreover, MUS-ASP enumerates at least twice as many MUSes compared to MARCO, ReMUS, and UNIMUS in 550, 394, and 446 instances, respectively. In contrast, there are $86,154,141$ instances where MARCO, ReMUS, and UNIMUS enumerates more MUSes than MUS-ASP, and the limitation is addressed in our hybrid approach discussed in **RQ3**.

	MARCO	ReMUS	UNIMUS	MUS-ASP
Rank	2.882	2.917	2.483	**1.576**

Table 1: Average ranks of MUS-ASP and other MUS enumerators; *the lower ranks are better*.

5.3 Analysis of RQ2.

To address **RQ2**, we compared the number of instances solved by MUS-ASP and CountMUST and their corresponding PAR2 scores. The statistics are shown in Table 2. We consider an instance solved by MUS-ASP if Clingo is able to enumerate all MUSes within the timeout. Their runtime performance is visually shown in Figure 1. Our experimental analysis reveals that MUS-ASP solved all instances that were solved by CountMUST and 38 more instances than CountMUST.

	UNIMUS	CountMUST	MUS-ASP
#Solved	545	887	**925**
PAR2	3630	1410	**1150**

Table 2: The performance comparison of MUS-ASP and existing MUS counters on MUS counting.

5.4 Analysis of RQ3.

We observe that the performance of MUS-ASP deteriorates with increasing formula size. Specifically, there are $85,92,92$ instances where MUS-ASP failed to enumerate any MUS, while MARCO, ReMUS, UNIMUS, respectively, succeeded in enumerating at least one MUS. To address this limitation, we developed a hybrid enumerator, denoted as X+MUS-ASP, for a given MUS enumerator X. The hybrid enumerator X+MUS-ASP invokes our prototype MUS-ASP for instances with fewer than 5000 clauses (a threshold chosen empirically) and defaults to the enumerator X for larger instances. We identified only two instances where X+MUS-ASP failed to enumerate any MUS, while the MUS enumerator X was able to enumerate at least one MUS. This indicates that the hybrid integration almost universally inherits the

[3]The extended version of this paper is available at: `https://arxiv.org/abs/2507.03929`.

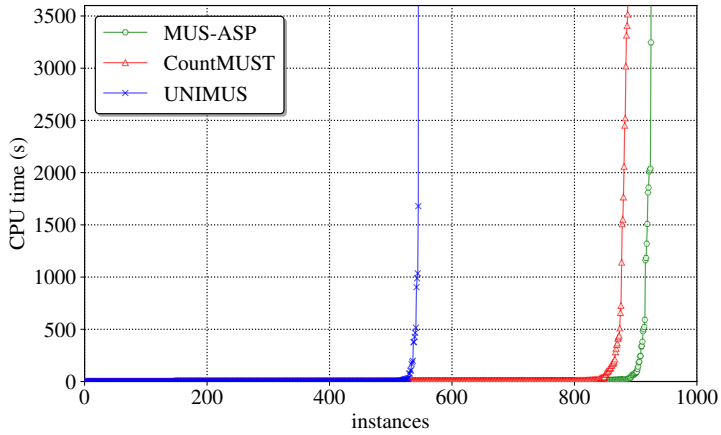


Figure 1: Runtime performance of MUS-ASP and existing MUS counters on MUS counting. A point (x, y) on the plot denotes that a tool counted x instances within y seconds.

better behavior of both MUS-ASP and the baseline X. The performance of the hybrid enumerator is reported in Table 3. The ranks demonstrate that integrating MUS-ASP with existing MUS enumerators substantially enhances overall performance.

	MARCO +MUS-ASP	ReMUS +MUS-ASP	UNIMUS +MUS-ASP	MUS-ASP
Rank	1.231	1.110	**1.053**	1.405

Table 3: Average ranks of hybrid MUS enumerators X+MUS-ASP. The hybrid approach uses MUS-ASP for instances having < 5000 clauses; otherwise, it defaults to the MUS enumerator X.

5.5 Further Experimental Analysis

We evaluate the effectiveness of different heuristics incorporated within the MUS-ASP framework. For MUS enumeration, the results are more nuanced. We found that all five heuristics perform optimally, whereas excluding $\mathbf{H}_5$ leads to significantly worse performance. In contrast, MUS counting yields clearer results, which are presented in Table 4. In the table, the leftmost column (no heuristic) reports the baseline performance of MUS-ASP without any heuristic applied — measured by the number of solved instances and PAR2 scores. The notation $\mathbf{H}_{i..j}$ $(i < j)$ indicates that the heuristics $\mathbf{H}_i$,...$\mathbf{H}_j$ are enabled in MUS-ASP. The Table 4 confirms that each heuristic, even when applied individually, contributes to enhanced performance in the MUS counting task.

	no heuristic	$\mathbf{H}_{1..2}$	$\mathbf{H}_{1..3}$	$\mathbf{H}_{1..5}$
#Solved	303	305	655	**927**
PAR2	5273	5261	2932	**1135**

Table 4: Performance of MUS-ASP on MUS counting benchmarks with different heuristic combinations.

Further experimental analysis is provided in the extended version.

6 Conclusion

This paper presented MUS-ASP, an ASP-based framework for MUS enumeration that encodes the problem as the computation of subset-minimal answer sets of a disjunctive logic program. Experimental results demonstrate that MUS-ASP outperforms state-of-the-art MUS enumeration and counting tools across a diverse set of benchmarks, with additional gains from heuristic optimizations and hybrid integration. As a direction for future work, we aim to develop approximate MUS counting techniques by leveraging ASP solvers that support XOR constraints [53].

References

[1] Durgesh Agrawal, Yash Pote & Kuldeep S Meel (2021): *Partition function estimation: A quantitative study*. *arXiv preprint arXiv:2105.11132*, doi:10.48550/arXiv.2105.11132.

[2] Mario Alviano, Carmine Dodaro, Salvatore Fiorentino, Alessandro Previti & Francesco Ricca (2022): *Enumeration of minimal models and MUSes in WASP*. In: *LPNMR*, Springer, pp. 29–42, doi:10.1007/978-3-031-15707-3_3.

[3] Mario Alviano, Carmine Dodaro, Salvatore Fiorentino, Alessandro Previti & Francesco Ricca (2023): *ASP and subset minimality: Enumeration, cautious reasoning and MUSes*. *Artificial Intell.* 320, p. 103931, doi:10.1016/j.artint.2023.103931.

[4] Mario Alviano, Carmine Dodaro, Nicola Leone & Francesco Ricca (2015): *Advances in WASP*. In: *LPNMR*, Springer, pp. 40–54, doi:10.1007/978-3-319-23264-5_5.

[5] Giovanni Amendola, Carmine Dodaro, Wolfgang Faber & Francesco Ricca (2021): *Paracoherent answer set computation*. *Artificial Intelligence* 299, p. 103519, doi:10.1016/j.artint.2021.103519.

[6] Zaher S Andraus, Mark H Liffiton & Karem A Sakallah (2007): *CEGAR-based formal hardware verification: A case study*. *Ann Arbor* 1001, p. 48.

[7] Fabrizio Angiulli, Rachel Ben-Eliyahu, Fabio Fassetti & Luigi Palopoli (2014): *On the tractability of minimal model computation for some CNF theories*. *Artificial Intelligence* 210, pp. 56–77, doi:10.1016/j.artint.2014.02.003.

[8] Fahiem Bacchus & George Katsirelos (2015): *Using minimal correction sets to more efficiently compute minimal unsatisfiable sets*. In: *CAV*, Springer, pp. 70–86, doi:10.1007/978-3-319-21668-3_5.

[9] James Bailey & Peter J Stuckey (2005): *Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization*. In: *PADL*, Springer, pp. 174–186, doi:10.1007/978-3-540-30557-6_14.

[10] Tomas Balyo, Marijn JH Heule & Matti Järvisalo (2017): *SAT Competition 2017–Solver and Benchmark Descriptions*. *SAT COMPETITION 2017*, pp. 14–15.

[11] Anton Belov, Marijn JH Heule & Joao Marques-Silva (2014): *MUS extraction using clausal proofs*. In: *SAT*, Springer, pp. 48–57, doi:10.1007/978-3-319-09284-3_5.

[12] Anton Belov & Joao Marques-Silva (2011): *Accelerating MUS extraction with recursive model rotation*. In: *FMCAD*, IEEE, pp. 37–40, doi:10.5555/2157654.2157663.

[13] Anton Belov & Joao Marques-Silva (2011): *Minimally unsatisfiable Boolean circuits*. In: *SAT*, Springer, pp. 145–158, doi:10.1007/978-3-642-21581-0_13.

[14] Anton Belov & Joao Marques-Silva (2012): *MUSer2: An efficient MUS extractor*. *JSAT* 8(3-4), pp. 123–128, doi:10.3233/SAT190094.

[15] Rachel Ben-Eliyahu & Rina Dechter (1994): *Propositional semantics for disjunctive logic programs*. *Annals of Mathematics and Artificial intelligence* 12, pp. 53–87, doi:10.1007/BF01530761.

[16] Jaroslav Bendík (2017): *Consistency checking in requirements analysis*. In: *ISSTA*, pp. 408–411, doi:10.1145/3092703.3098239.

[17] Jaroslav Bendík, Nikola Benes, Ivana Cerná & Jiri Barnat (2016): *Tunable online MUS/MSS enumeration*.

[18] Jaroslav Bendík & Ivana Černá (2020): *MUST: minimal unsatisfiable subsets enumeration tool*. In: *TACAS*, Springer, pp. 135–152, doi:10.1007/978-3-030-45190-5_8.

[19] Jaroslav Bendík & Ivana Černá (2020): *Replication-guided enumeration of minimal unsatisfiable subsets*. In: *CP*, Springer, pp. 37–54, doi:10.1007/978-3-030-58475-7_3.

[20] Jaroslav Bendík, Ivana Černá & Nikola Beneš (2018): *Recursive online enumeration of all minimal unsatisfiable subsets*. In: *ATVA*, Springer, pp. 143–159, doi:10.1007/978-3-030-01090-4_9.

[21] Jaroslav Bendík & Kuldeep S Meel (2020): *Approximate counting of minimal unsatisfiable subsets*. In: *CAV*, Springer, pp. 439–462, doi:10.1007/978-3-030-53288-8_21.

[22] Jaroslav Bendík & Kuldeep S Meel (2021): *Counting minimal unsatisfiable subsets*. In: *CAV*, Springer, pp. 313–336, doi:10.1007/978-3-030-81688-9_15.

[23] Gerhard Brewka, James Delgrande, Javier Romero & Torsten Schaub (2015): *asprin: Customizing answer set preferences without a headache*. In: *AAAI*, 29, doi:10.1609/aaai.v29i1.9398.

[24] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca & Torsten Schaub (2020): *ASP-Core-2 input language format*. *TPLP* 20(2), pp. 294–309, doi:10.1017/S1471068419000450.

[25] Supratik Chakraborty, Kuldeep S Meel & Moshe Y Vardi (2013): *A scalable approximate model counter*. In: *CP*, Springer, pp. 200–216, doi:10.1007/978-3-642-40627-0_18.

[26] Huan Chen & Joao Marques-Silva (2012): *Improvements to satisfiability-based boolean function bi-decomposition*. In: *VLSI-SoC*, pp. 52–72, doi:10.1007/978-3-642-32770-4_4.

[27] Zhi-Zhong Chen & Seinosuke Toda (1995): *The complexity of selecting maximal solutions*. *Information and Computation* 119(2), pp. 231–239, doi:10.1109/SCT.1993.336516.

[28] Keith L Clark (1978): *Negation as failure*. *Logic and data bases*, pp. 293–322, doi:10.1007/978-1-4684-3384-5_11.

[29] Orly Cohen, Moran Gordon, Michael Lifshits, Alexander Nadel & Vadim Ryvchin (2010): *Designers work less with quality formal equivalence checking*. In: *DVCon*.

[30] Johan De Kleer & Brian C Williams (1987): *Diagnosing multiple faults*. *Artificial Intelligence* 32(1), pp. 97–130, doi:10.1016/0004-3702(87)90063-4.

[31] Emanuele Di Rosa, Enrico Giunchiglia & Marco Maratea (2010): *Solving satisfiability problems with preferences*. *Constraints* 15, pp. 485–515, doi:10.1007/s10601-010-9095-y.

[32] Carmine Dodaro, Philip Gasteiger, Kristian Reale, Francesco Ricca & Konstantin Schekotihin (2019): *Debugging non-ground ASP programs: Technique and graphical tools*. *TPLP* 19(2), pp. 290–316, doi:10.1017/S1471068418000492.

[33] Arnaud Durand, Miki Hermann & Phokion G Kolaitis (2005): *Subtractive reductions and complete problems for counting complexity classes*. *Theoretical Computer Science* 340(3), pp. 496–513, doi:10.1016/j.tcs.2005.03.012.

[34] Thomas Eiter & Georg Gottlob (1995): *On the computational cost of disjunctive logic programming: Propositional case*. *Annals of Mathematics and Artificial Intelligence* 15, pp. 289–323, doi:10.1007/BF01536399.

[35] Esra Erdem, Michael Gelfond & Nicola Leone (2016): *Applications of answer set programming*. *AI Magazine* 37(3), pp. 53–68, doi:10.1609/aimag.v37i3.2678.

[36] Johannes K Fichte & Stefan Szeider (2015): *Backdoors to normality for disjunctive logic programs*. *TOCL* 17(1), pp. 1–23, doi:10.1145/2818646.

[37] Zhaohui Fu & Sharad Malik (2006): *On solving the partial MaxSAT problem*. In: *SAT*, Springer, pp. 252–265, doi:10.1007/11814948_25.

[38] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub & Philipp Wanko (2016): *Theory solving made easy with Clingo 5*. In: *ICLP*, pp. 2–1, doi:10.4230/OASIcs.ICLP.2016.2.

[39] Martin Gebser, Benjamin Kaufmann, Javier Romero, Ramón Otero, Torsten Schaub & Philipp Wanko (2013): *Domain-specific heuristics in answer set programming*. In: *AAAI*, 27, pp. 350–356, doi:10.1613/jair.1.14091.

[40] Martin Gebser, Benjamin Kaufmann & Torsten Schaub (2012): *Conflict-driven answer set solving: From theory to practice*. *Artificial Intelligence* 187, pp. 52–89, doi:10.1016/j.artint.2012.04.001.

[41] Michael Gelfond & Vladimir Lifschitz (1991): *Classical negation in logic programs and disjunctive databases*. *New Generation Computing* 9, pp. 365–385, doi:10.1007/BF03037169.

[42] Dimitrios Gunopulos, Roni Khardon, Heikki Mannila, Sanjeev Saluja, Hannu Toivonen & Ram Sewak Sharma (2003): *Discovering all most specific sentences*. *ACM Trans. on Database Systems* 28(2), pp. 140–174, doi:10.1145/777943.777945.

[43] Benjamin Han & Shie-Jue Lee (1999): *Deriving minimal conflict sets by CS-trees with mark set in diagnosis from first principles*. *IEEE Trans. on Sys., Man, and Cyber.* 29(2), pp. 281–286, doi:10.1109/3477.752801.

[44] Anthony Hunter & Sébastien Konieczny (2008): *Measuring Inconsistency through Minimal Inconsistent Sets*. *KR* 8(358-366), p. 42.

[45] Antonio Ielo, Giuseppe Mazzotta, Francesco Ricca & Rafael Peñaloza (2025): *Towards ASP-based minimal unsatisfiable cores enumeration for LTLf*. In: *CEUR WORKSHOP*, 3904, pp. 49–55.

[46] Alexey Ignatiev, Alessandro Previti, Mark Liffiton & Joao Marques-Silva (2015): *Smallest MUS extraction with minimal hitting set dualization*. In: *CP*, Springer, pp. 173–182, doi:10.1007/978-3-319-23219-5_13.

[47] Alexander Ivrii, Sharad Malik, Kuldeep S Meel & Moshe Y Vardi (2016): *On computing minimal independent support and its applications to sampling and counting*. *Constraints* 21, pp. 41–58, doi:10.1007/s10601-015-9204-z.

[48] Dietmar Jannach & Thomas Schmitz (2016): *Model-based diagnosis of spreadsheet programs: a constraint-based debugging approach*. *Automated Software Engineering* 23, pp. 105–144, doi:10.1007/s10515-014-0141-7.

[49] Mikoláš Janota, Radu Grigore & Joao Marques-Silva (2010): *Counterexample guided abstraction refinement algorithm for propositional circumscription*. In: *JELIA*, Springer, pp. 195–207, doi:10.1007/978-3-642-15675-5_18.

[50] Md Mohimenul Kabir, Supratik Chakraborty & Kuldeep S Meel (2025): *Counting Answer Sets of Disjunctive Answer Set Programs*. *TPLP*, pp. 1–19, doi:10.1017/S1471068425100124.

[51] Mohimenul Kabir (2024): *Minimal model counting via knowledge compilation*. *arXiv preprint*. arXiv:2409.10170.

[52] Mohimenul Kabir, Supratik Chakraborty & Kuldeep S Meel (2024): *Exact ASP counting with compact encodings*. In: *AAAI*, 38, pp. 10571–10580, doi:10.1609/aaai.v38i9.28927.

[53] Mohimenul Kabir, Flavio O Everardo, Ankit K Shukla, Markus Hecher, Johannes Klaus Fichte & Kuldeep S Meel (2022): *ApproxASP–a scalable approximate answer set counter*. In: *AAAI*, 36, pp. 5755–5764, doi:10.1609/aaai.v36i5.20518.

[54] Mohimenul Kabir & Kuldeep S Meel (2023): *A Fast and Accurate ASP Counting Based Network Reliability Estimator.* In: *LPAR*, 94, pp. 270–287, doi:10.29007/kc6q.

[55] Mohimenul Kabir & Kuldeep S Meel (2024): *On Lower Bounding Minimal Model Count*. *TPLP* 24(4), pp. 586–605, doi:10.1017/S147106842400036X.

[56] Mohimenul Kabir, Van-Giang Trinh, Samuel Pastva & Kuldeep S Meel (2025): *Scalable Counting of Minimal Trap Spaces and Fixed Points in Boolean Networks*. doi:10.48550/arXiv.2506.06013.

[57] George Katsirelos & Fahiem Bacchus (2016): *Finding a collection of MUSes incrementally*. In: *CPAIOR*, Springer, pp. 35–44, doi:10.1007/978-3-319-33954-2_3.

[58] Hans Kleine Büning & Oliver Kullmann (2009): *Minimal unsatisfiability and autarkies*. In: *Handbook of Satisfiability*, IOS Press, pp. 339–401, doi:10.3233/978-1-58603-929-5-339.

[59] Oliver Kullmann & Joao Marques-Silva (2015): *Computing maximal autarkies with few and simple oracle queries*. In: *SAT*, Springer, pp. 138–155, doi:10.1007/978-3-319-24318-4_11.

[60] Anna L.D. Latour, Arunabha Sen & Kuldeep S. Meel (2023): *Solving the Identifying Code Set Problem with Grouped Independent Support*. In: *IJCAI*, pp. 1971–1978, doi:10.24963/ijcai.2023/219.

[61] Yuliya Lierler (2005): *Cmodels–SAT-based disjunctive answer set solver*. In: *LPNMR*, Springer, pp. 447–451, doi:10.1007/11546207_44.

[62] Mark H Liffiton & Ammar Malik (2013): *Enumerating infeasibility: Finding multiple MUSes quickly*. In: *CPAIOR*, Springer, pp. 160–175, doi:10.1007/978-3-642-38171-3_11.

[63] Mark H Liffiton, Michael D Moffitt, Martha E Pollack & Karem A Sakallah (2005): *Identifying conflicts in overconstrained temporal problems*. In: *IJCAI*, pp. 205–211.

[64] Mark H Liffiton, Alessandro Previti, Ammar Malik & Joao Marques-Silva (2016): *Fast, flexible MUS enumeration*. *Constraints* 21, pp. 223–250, doi:10.1007/s10601-015-9183-0.

[65] Mark H Liffiton & Karem A Sakallah (2008): *Algorithms for computing minimal unsatisfiable subsets of constraints*. *Journal of Automated Reasoning* 40, pp. 1–33, doi:10.1007/s10817-007-9084-z.

[66] Panagiotis Lymperopoulos & Liping Liu (2024): *Graph Pruning for Enumeration of Minimal Unsatisfiable Subsets*. In: *AISTATS*, PMLR, pp. 2647–2655.

[67] Victor W Marek & Miroslaw Truszczyński (1999): *Stable models and an alternative logic programming paradigm*. In: *The Logic Programming Paradigm*, Springer, pp. 375–398, doi:10.1007/978-3-642-60085-2_17.

[68] Wiktor Marek & Mirosław Truszczyński (1991): *Autoepistemic logic*. *Journal of the ACM (JACM)* 38(3), pp. 587–618, doi:10.1145/116825.116836.

[69] Joao Marques-Silva, Alexey Ignatiev, António Morgado, Vasco Manquinho & Inês Lynce (2014): *Efficient autarkies*. In: *ECAI*, IOS Press, pp. 603–608, doi:10.3233/978-1-61499-419-0-603.

[70] Joao Marques-Silva & Mikolás Janota (2014): *On the query complexity of selecting few minimal sets*. In: *ECCC*, 21, doi:10.1016/j.artint.2016.01.002.

[71] Joao Marques-Silva & Jordi Planes (2008): *Algorithms for maximum satisfiability using unsatisfiable cores*. In: *DATE*, pp. 408–413, doi:10.1145/1403375.1403474.

[72] Carlos Mencía, Oliver Kullmann, Alexey Ignatiev & Joao Marques-Silva (2019): *On computing the union of MUSes*. In: *SAT*, Springer, pp. 211–221, doi:10.1007/978-3-030-24258-9_15.

[73] Burkhard Monien & Ewald Speckenmeyer (1985): *Solving satisfiability in less than 2^n steps*. *Discrete Applied Mathematics* 10(3), pp. 287–295, doi:10.1016/0166-218X(85)90050-2.

[74] Antonio Morgado, Mark Liffiton & Joao Marques-Silva (2013): *MaxSAT-based MCS enumeration*. In: *HVC*, Springer, pp. 86–101, doi:10.1007/978-3-642-39611-3_13.

[75] Kedian Mu, Weiru Liu & Zhi Jin (2011): *A general framework for measuring inconsistency through minimal inconsistent sets*. *Knowledge and Information Systems* 27, pp. 85–114, doi:10.1007/s10115-010-0295-y.

[76] Alexander Nadel (2010): *Boosting minimal unsatisfiable core extraction*. In: *FMCAD*, IEEE, pp. 221–229, doi:10.5555/1998496.1998537.

[77] Alexander Nadel, Vadim Ryvchin & Ofer Strichman (2014): *Accelerated deletion-based extraction of minimal unsatisfiable cores*. *JSAT* 9(1), pp. 27–51, doi:10.3233/SAT190100.

[78] Nina Narodytska, Nikolaj Bjørner, Maria Cristina Marinescu & Mooly Sagiv (2018): *Core-guided minimal correction set and core enumeration*. In: *IJCAI*, IJCAI, pp. 1353–1361, doi:10.24963/ijcai.2018/188.

[79] Max Ostrowski, Loïc Paulevé, Torsten Schaub, Anne Siegel & Carito Guziolowski (2016): *Boolean network identification from perturbation time series data combining dynamics abstraction and logic programming*. *Biosystems* 149, pp. 139–153, doi:10.1016/j.biosystems.2016.07.009.

[80] David Rajaratnam, Torsten Schaub, Philipp Wanko, Kai Chen, Sirui Liu & Tran Cao Son (2023): *Solving an industrial-scale warehouse delivery problem with answer set programming modulo difference constraints*. *Algorithms* 16(4), p. 216, doi:10.3390/a16040216.

[81] Raymond Reiter (1987): *A theory of diagnosis from first principles*. *Artificial intelligence* 32(1), pp. 57–95, doi:10.1016/0004-3702(87)90062-2.

[82] Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura & David L Dill (2018): *Learning a SAT solver from single-bit supervision*. *arXiv preprint*. arXiv:1802.03685.

[83] Emanuel Sperner (1928): *Ein satz über untermengen einer endlichen menge*. *Mathematische Zeitschrift* 27(1), pp. 544–548, doi:10.1007/BF01171114.

[84] Roni Stern, Meir Kalech, Alexander Feldman & Gregory Provan (2012): *Exploring the duality in conflict-directed model-based diagnosis*. In: *AAAI*, 26, pp. 828–834, doi:10.1609/aaai.v26i1.8231.

[85] Peter J Stuckey, Martin Sulzmann & Jeremy Wazny (2003): *Interactive type debugging in Haskell*. In: *ACM SIGPLAN workshop on Haskell*, pp. 72–83, doi:10.1145/871895.871903.

[86] Giang Trinh, Belaid Benhamou, Samuel Pastva & Sylvain Soliman (2024): *Scalable Enumeration of Trap Spaces in Boolean Networks via Answer Set Programming*. In: *AAAI*, 38, pp. 10714–10722, doi:10.1609/aaai.v38i9.28943.

[87] Siert Wieringa (2012): *Understanding, improving and parallelizing MUS finding using model rotation*. In: *CP*, Springer, pp. 672–687, doi:10.1007/978-3-642-33558-7_49.

Machine Learning Model Integration with Open World Temporal Logic for Process Automation

Dyuman Aditya
Syracuse University, Syracuse, NY USA
daditya@syr.edu

Colton Payne
Arizona State University, Tempe, AZ USA
crpayne2@asu.edu

Mario Leiva
Dept. of Computer Science & Eng.,
Inst. for Computer Science & Eng.
Universidad Nacional del Sur, BA, Argentina
mario.leiva@cs.uns.edu.ar

Paulo Shakarian
Syracuse University, Syracuse, NY USA
pashakar@syr.edu

Recent advances in Machine Learning (ML) have produced models that extract structured information from complex data. However, a significant challenge lies in translating these perceptual or extractive outputs into actionable and explainable decisions within complex operational workflows. To address these challenges, this paper introduces a novel approach that integrates the outputs of various machine learning models directly with the PyReason framework, an open-world temporal logic programming reasoning engine. PyReason's foundation in generalized annotated logic allows for the incorporation of real-valued outputs (e.g., probabilities, confidence scores) from a diverse set of ML models, treating them as truth intervals within its logical framework. Crucially, PyReason provides mechanisms, implemented in Python, to continuously poll ML model outputs, convert them into logical facts, and dynamically recompute the minimal model to enable decision-making in real-time. Furthermore, its native support for temporal reasoning, knowledge graph integration, and fully explainable interface traces enables an analysis of time-sensitive process data and existing organizational knowledge. By combining the strengths of perception and extraction from ML models with the logical deduction and transparency of PyReason, we aim to create a powerful system for automating complex processes. This integration is well suited for use cases in numerous domains, including manufacturing, healthcare, and business operations.

1 Introduction

Recent advances in Machine Learning (ML) have produced models that extract structured information from complex data. Object detectors, image classifiers, and large language models can now recognize patterns, classify inputs, and extract entities. Their outputs, such as object locations, image labels, or text sentiments, can help automate and improve real-world processes.

However, a significant challenge lies in translating these perceptual or extractive outputs into actionable and explainable decisions within complex operational workflows. ML models, while proficient in their specific tasks, often lack the inherent mechanisms for explainable logical reasoning, transparently managing uncertainty, integrating explicit domain knowledge, or handling the temporal dynamics crucial for many automation scenarios. Bridging this gap requires a framework that can synthesize these varied ML outputs and subject them to rigorous, explainable logical analysis.

To address these challenges, this paper introduces a novel approach that integrates the outputs of various machine learning models directly with the PyReason framework [2], an open-world temporal logic reasoning engine. Figure 1 provides a high-level overview of this integration framework. PyReason's

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 34–46, doi:10.4204/EPTCS.439.4

© D. Aditya, C. Payne, M. Leiva, P. Shakarian
This work is licensed under the
Creative Commons Attribution License.

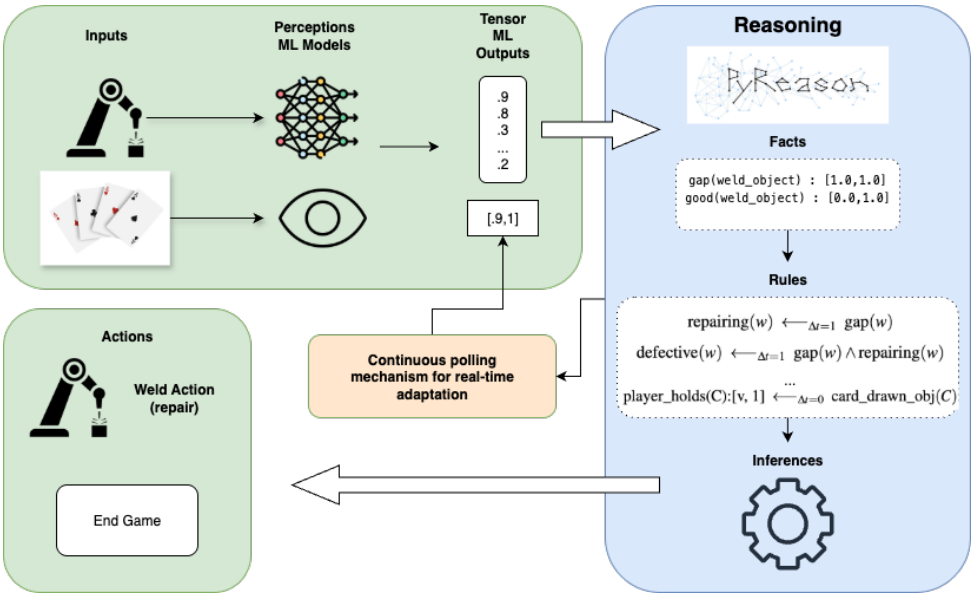


Figure 1: A conceptual overview of the proposed integration framework. (1) Machine learning models provide perceptual outputs (e.g., tensor, probabilities). (2) The PyReason framework converts these outputs into logical facts with truth-value intervals. (3) These facts are used within a temporal logic program to derive actionable, reasoned decisions. The feedback loop illustrates the continuous polling mechanism for real-time adaptation

foundation in generalized annotated logic [7] directly incorporates real-valued outputs (e.g., probabilities, confidence scores) from diverse ML models, treating them as truth intervals within its logical framework. We note that the existing work focuses on integration between machine learning models and logic-based reasoning by either modifying the ML model (e.g., see [4, 10, 12, 5, 6, 3]) or by adding rules about the failure of the model to the logic program (e.g., see [8, 1, 9]). In this paper, we examine practical methods complementary to both approaches - focusing on software engineering questions concerning the integration between logic programs and ML models. This distinction carries a significant impact: by not requiring modifications to the ML model, our approach allows for the seamless integration of pre-trained, "black-box" systems. This enhances modularity and practical applicability, as it enables the use of state-of-the-art proprietary models or those that are computationally expensive to fine-tune, thus lowering the barrier for creating complex neuro-symbolic applications.

A central pillar of our proposal is the ability of PyReason to implement logic programs with Python code. This is achieved through mechanisms that enable Python code to continuously poll ML model results, converting these perceptions into logical facts that are directly injected into the logic program. This process not only feeds the reasoning engine, but also triggers the re-computation of the minimal model, ensuring that logical decisions always reflect the latest observations from the ML model. Given the ICLP audience's familiarity with both Python and logic programming, we have prioritized a design that explicitly intersperses both paradigms, facilitating an understanding of the interaction between ML perception and logical deduction. We have made several design decisions to simplify this integration and improve usability. For example, the introduction of specific base classes such as `LogicIntegrationBase` and `TemporalLogicIntegratedClassifier` standardizes the inference flow and the conversion of predictions into logical facts. These decisions simplify syntax, manage optional arguments, and support a

robust, modular approach to complex process automation.

In addition to this integration, PyReason supports temporal reasoning, knowledge graph integration, and explainable inference traces. While traditional logic programming languages like Prolog can provide explainable reasoning paths, they are less suited for integrating existing knowledge graphs and handling time-based logic. Python, by contrast, offers better support for these needs through libraries like GraphML and NetworkX, and is more adaptable for implementing temporal reasoning. By combining outputs from machine learning models with PyReason's logical reasoning capabilities, we aim to build a system for automating complex processes. This approach is applicable across domains such as manufacturing (using object detection for quality control), healthcare (image classification for diagnostics), and business operations (LLMs for document processing and customer support).

The remainder of this paper is structured as follows. Section2 introduces the PyReason framework, its logical foundations, and its reasoning capabilities. Section3 details the integration with machine learning models via a standardized interface. Section4 presents two case studies—robotic welding and a card game—to illustrate practical applications. Section5 concludes with a discussion of the broader implications.

2 The PyReason Framework

PyReason is a software framework developed for open-world temporal logic, implemented in Python [2]. It is founded on generalized annotated logic, a system which not only encapsulates the current range of differentiable (i.e., fuzzy) first-order logics but also integrates temporal extensions to support inference over finite durations. A primary design objective of PyReason is to provide direct support for reasoning on graphical structures, such as knowledge graphs, social networks, and biological networks. The framework is engineered to generate fully explainable traces of its inference processes, ensuring transparency. It also includes type-checking and a memory-efficient design for scalable deductive inference. PyReason endeavors to supply a modular and extensible platform for deduction, addressing the constraints of bespoke software frequently developed for particular logics within neuro-symbolic frameworks. It captures a breadth of capabilities observed in diverse neuro-symbolic frameworks, encompassing fuzzy logic, open-world reasoning, temporal reasoning, and graph-based reasoning. The PyReason software is publicly accessible via GitHub (`https://github.com/lab-v2/pyreason`).

2.1 Core Logical Principles and Reasoning over Graphs

The logical underpinning of PyReason is generalized annotated logic, augmented with various extensions to support a wide array of capabilities. A defining characteristic is its application of real-valued interval annotations, enabling logical statements to be annotated with elements from a lower semilattice structure composed of intervals that are subsets of $[0,1]$. This design aligns with truth intervals in fuzzy operators and paradigms in neuro-symbolic reasoning. Classical logic supports annotations $[0,0]$ (false) and $[1,1]$ (true), while tri-valued logic can use $[0,1]$ for "unknown". Negation of an atom annotated with $[l,u]$ results in an annotation of $[1-u,1-l]$ for its strong negation.

PyReason integrates temporal logic over finite periods through "interpretations", which are an initial set of facts correct at $t=0$. Temporal annotations $I(A,\hat{t}) : [L,U]$ represent discrete time-points, where $\hat{t}$ is a specific time t or s for static interpretations (unchanging over time). Non-static predicates can change annotations over time without a monotonicity requirement. The framework operates under an open-world assumption: interpretations not explicitly defined are initially "unknown" (e.g., $[0,1]$).

The framework is designed to reason about graphical structures, such as knowledge graphs. It assumes a graph $G = (\mathscr{C}, E)$, where nodes $\mathscr{C}$ are constants, and edges E specify potential relationships. Predicates are typically unary (node labels) or binary (edge labels). A reserved binary predicate 'rel' exists, where $rel(a,b)$ is a tautology if $(a,b) \in E$ and uncertain otherwise. Graphical syntactic extensions allow for operators like requiring the existence of k items.

2.2 Key Reasoning Capabilities and Features

Logical rules are the primary mechanism for modifying annotations of non-static predicates. PyReason supports ground rules, universally quantified non-ground rules (for reasoning within or across edges), and rules with rule-based quantifiers in the head. Rules incorporate a temporal component through a time delay, denoted as Δt. Syntactically, this delay is represented as a subscript to the implication arrow (e.g., $\leftarrow_{\Delta t}$, specifying the number of timesteps that must elapse after the body is satisfied before the head's annotation is updated. This feature explicitly defines the temporal nature of the logic programs used within PyReason. Rule heads often involve functions $f(x_1, ..., x_n)$ (e.g., T-norms, T-conorms, algebraic functions) to combine annotations from body literals.

Deduction centers around a fixpoint operator, Γ, proven to produce all entailed atoms by a logic program. This computation is exact, yielding the minimal model and allowing for entailment checking. A significant advantage is the fully explainable result, with PyReason providing the series of inference steps for any entailment query.

To mitigate computational complexity, especially the grounding problem, PyReason incorporates predicate-constant type checking. This leverages knowledge graph sparsity, where nodes have types and predicates are defined over specific types. During initialization, ground atoms are created only for compatible predicate-constant pairs, significantly reducing the search space. This is an optional feature.

PyReason detects and manages logical inconsistencies. Inconsistency can occur if a new interpretation $[L', U']$ is not a subset of the current $[L, U]$, or between an atom and its negation, or complementary predicates. PyReason flags and can report these inconsistencies. The explainable fixpoint operator allows tracing the cause. An option exists to resolve inconsistency by resetting the annotation to $[0, 1]$ (uncertainty) and setting the atom to static.

2.3 Implementation and Neuro-Symbolic Relevance

PyReason is a modern Python framework designed for scalability and correctness. Graphical input is via PrologML or a Networkx Prolog, with initial conditions and rules in text format. The Numba JIT compiler optimizes key operations for performance and supports CPU parallelism. For memory efficiency, only current interpretations are stored. However, past states are reconstructible via rule traces, which enable explainability for long-running implementations. This design supports canonical and non-canonical models with minimal computational overhead by recomputing interpretations at each step.

PyReason is strategically positioned as an instrument in neuro-symbolic AI. Generalized annotated logic, its foundation, captures a wide spectrum of real-valued, temporal, and fuzzy logics used in frameworks like Logical Tensor Networks (LTN) and Logical Neural Networks (LNN), both viewable as subsets of annotated logic. PyReason can conduct inference on logic programs learned or modified by such systems. It enables capabilities like graph-based and temporal reasoning, potentially absent in other neuro-symbolic frameworks' native logics. The explainability of its inference is essential to understand and debug complex neuro-symbolic models. By supporting generalized annotated logic and its extensions, PyReason facilitates system design independent of the learning process, offering a robust deductive

engine.

3 Integration with Machine Learning Models

PyReason allows users to integrate machine learning (ML) models directly within the logical framework. ML models output *tensors*, which are converted into PyReason *facts* that can be used to perform inference. This section outlines this process and illustrates its simplicity within our framework.

3.1 Logic Integrated Base Class

PyReason defines an abstract base class `LogicIntegrationBase`, designed for subclassing, to integrate specific machine learning models. This class declares three protected methods that are called sequentially and must be implemented by every subclass:

1. `_infer`: Execute the model and return its raw output tensor.
2. `_postprocess`: Transform the tensor (e.g. apply a sigmoid) into interpretable scores.
3. `_pred_to_facts`: Convert the post-processed predictions into logical facts for downstream reasoning.

3.2 Logic Integrated Classifier

This section covers how the `LogicIntegrationBase` class can be used to integrate a simple binary classifier which is then used in the example in Subsection 4.1. We outline the implementation of the three methods discussed earlier.

Model inference: The `_infer` method is implemented as follows, where the underlying model is called to get the raw logits.

Listing 1: Model Inference

```
    def _infer(self, x: torch.Tensor) -> torch.Tensor:
        return self.model(x)
```

Post Processing The `_postprocess` method applies a sigmoid function over the raw model outputs to get a probability distribution over the classes.

Listing 2: Post-Processing

```
    def _postprocess(self, raw_output: torch.Tensor) -> torch.Tensor:
        logits = raw_output  # Output from the _infer function
        return F.softmax(logits, dim=1)
```

Predictions to Facts The `_pred_to_facts` method is responsible for converting the post-processed predictions into logical facts. To do this, we define a set of user interface options:

1. `threshold`: Beyond which predictions should be integrated
2. `set_lower_bound`: Whether to modify the lower bound of the resulting fact
3. `set_upper_bound`: Whether to modify the upper bound of the resulting fact
4. `snap_value`: The value to set the upper or lower bound (depending on the setting) of the resulting fact. If this is left empty, the original output from the post-processing function will be used.

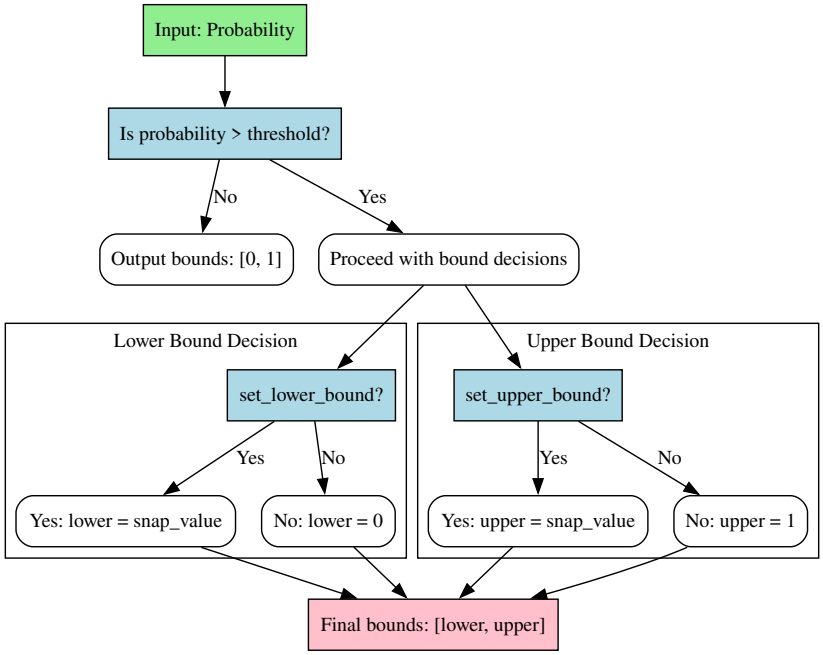


Figure 2: Conversion from probabilities to annotated bounds

For simplicity, we outline the process of converting probabilities to annotated bounds as a flowchart depicted in Figure 2. By following this logic, we are able to convert model output tensors into annotated facts which can be added into PyReason for downstream inference.

3.3 Automated Temporal Logic Integrated Classifier

In many real-world applications, data arrives continuously from sensors or cameras. Logical inferences must adapt in real time. To address this, PyReason provides the `TemporalLogicIntegratedClassifier`, a subclass of `LogicIntegratedClassifier` that ties PyReason's internal temporal reasoning to wall-clock time. This class adds two key parameters: `poll_interval`, which specifies the frequency (in seconds or PyReason time) to poll the model with new input data if the `poll_condition` is either satisfied or not defined; and `poll_condition`, a PyReason query that determines whether a new inference cycle should be triggered.

At each interval, PyReason evaluates the `poll_condition`. If the condition is satisfied, or if no condition is specified, the system automatically:

1. Loads new input (e.g. an image or sensor reading).
2. Invokes the underlying ML model to produce raw outputs.
3. Converts those outputs into annotated facts (as shown in Section 3).
4. Runs the fixpoint operator to update the interpretation given the new facts.

This end-to-end loop (Figure 3) requires no additional glue code: PyReason handles the timing, model polling, fact injection, and inference in a single, explainable pipeline.

Section 4 illustrates two concrete workflows built on this machinery: defect detection in robotic welding and decision-making in a simulated card game. In each case, the temporal classifier ensures

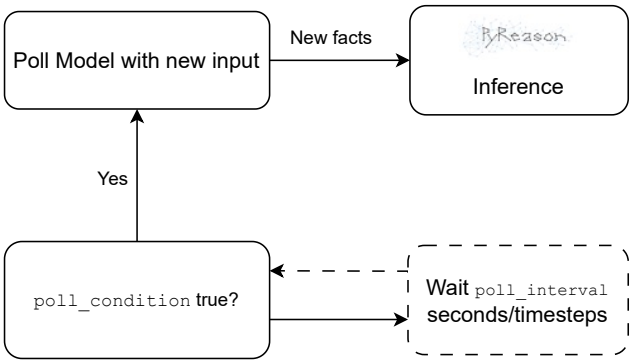


Figure 3: Background model-polling and inference loop

that as soon as new data are available, the logic program reflects the latest observations and derives the appropriate conclusions.

4 Illustrative Case Studies

To demonstrate the practical application and versatility of integrating machine learning models within the PyReason framework, this section presents two distinct case studies. The first, a robotic welding scenario, illustrates how the system can be applied to industrial automation for real-time quality control. It showcases the use of a temporal logic-integrated classifier to detect weld defects, trigger an automated repair process, and make a final determination of product quality. The second case study, a simple card game, highlights PyReason's ability to manage uncertainty and guide decision-making over time by periodically integrating outputs from an image classifier and applying annotated logic to the evolving state of the game.

4.1 Welding Classifier

In an industrial robotic welding cell, detecting weld defects in real time is essential to maintaining product quality and avoiding costly rework. Here, we apply a pre-trained weld-defect detector to identify gaps in each weld image, feed those results into PyReason to attempt automated gap closure, and mark the part as defective if the gap cannot be closed.

4.1.1 Logical Encoding in PyReason

We use the automated temporal logic integrated classifier described in Section 3.3 to integrate a welding defect detection model with a logic program that consists of two simple rules:

$$\text{repairing}(w) \longleftarrow_{\Delta t=1} \text{gap}(w)$$

$$\text{defective}(w) \longleftarrow_{\Delta t=1} \text{gap}(w) \wedge \text{repairing}(w)$$

The `gap(weld_object)` atom is added to the program by PyReason's `LogicIntegratedClassifier`. These two rules help determine if the weld was unsuccessful and if after a repair was attempted there is

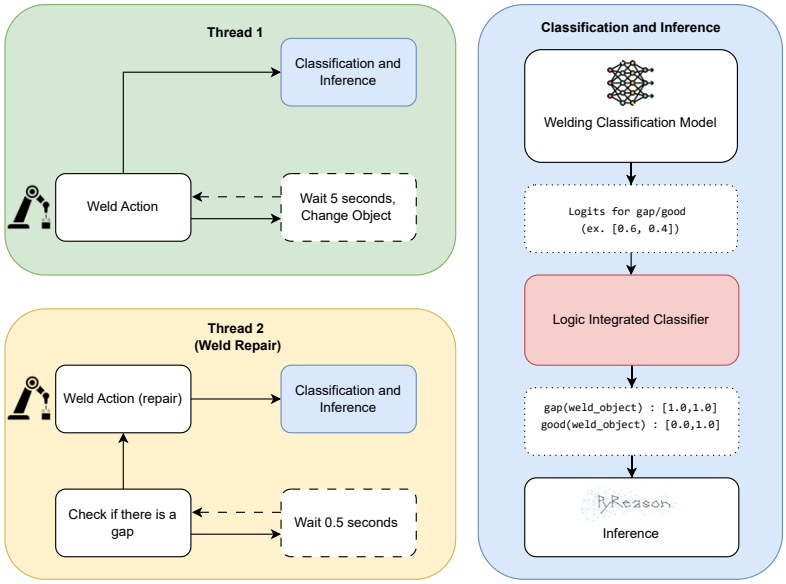


Figure 4: Welding example flowchart

still a gap, then the part is marked as defective. A flowchart of the pipeline can be seen in Figure 4. The repair and conclusion that a part might be defective run in a background routine (Thread 2) while the main thread (Thread 1) goes over each object to be welded and performs the welding task.

We use the `TemporalLogicIntegratedClassifier` defined in Section 3.3, with a `poll_interval` set to 0.5 seconds and the `poll_condition` set to `gap(weld_object):[1,1]`. This allows us to check whether there is a gap in the weld every 0.5 seconds and attempt to repair the gap in the background.

4.1.2 Interpreting the Inference Trace

Table 1 showcases an explainable PyReason trace after welding 5 objects over 7 fixed-point operations (FPO). It can be seen that both thread 1 and thread 2 update the interpretation simultaneously. Thread 1 is always called first on every new object, and thread 2 is invoked only when a repair is necessary. In FPOs 2-3, a repair is attempted due to a gap being detected in FPO 1 and the object is deemed defective after an unsuccessful repair. Similarly, another repair is invoked in FPO 5, but the repair is successful and the part is not marked as defective.

4.2 Simple Card Game

This case study demonstrates how PyReason's open-world temporal logic can drive decision-making in a simple "42" card game by periodically integrating the output from an image classifier and applying annotated logic to manage uncertainty over time. The objective of the game is to draw as many cards as possible without exceeding a point total of 42. Card values are defined as 3 points for an Ace, 6 for numeric cards, and 9 for face cards. At each turn, a decision is made to draw another card if there is a nonzero probability of remaining at or below the 42-point limit, based on the cards remaining in the deck. The game concludes when any subsequent draw would guarantee exceeding the limit.

Table 1: PyReason explainable trace for the welding scenario

FPO	Node	Label	Old Bound	New Bound	Thread
0	weld_object	good	[0.0,1.0]	[1.0,1.0]	1
1	weld_object	gap	[0.0,1.0]	[1.0,1.0]	1
2	weld_object	repairing	[0.0,1.0]	[1.0,1.0]	2
2	weld_object	gap	[0.0,1.0]	[1.0,1.0]	2
3	weld_object	defective	[0.0,1.0]	[1.0,1.0]	2
4	weld_object	gap	[0.0,1.0]	[1.0,1.0]	1
5	weld_object	repairing	[0.0,1.0]	[1.0,1.0]	2
5	weld_object	good	[0.0,1.0]	[1.0,1.0]	2
6	weld_object	good	[0.0,1.0]	[1.0,1.0]	1
7	weld_object	good	[0.0,1.0]	[1.0,1.0]	1

4.2.1 Logical Encoding in PyReason

We encode the game state using five predicates and three rules:

Predicates

- `deck_holds(C, D)`: links each card C with the deck D as edges in a graph.
- `card_drawn_obj(C)`: classifier output indicating that card C was drawn.
- `player_holds(C)`: assigns a fuzzy interval score to C based on `card_drawn_obj(C)`.
- `hand_as_point_vals(V)`: total value of held cards, represented as a fuzzy interval.
- `odds_of_losing(P)`: probability that the next draw exceeds 42 points.

Rules

$$\text{player_holds(C):[v, 1]} \longleftarrow_{\Delta t=0} \text{card_drawn_obj}(C)$$

$$\text{hand_as_point_vals(hand)} : \text{append_hand_annotation_fn} \longleftarrow_{\Delta t=0} \text{player_holds(card)} : [0.3, 1]$$

$$\text{odds_of_losing(hand)} : \text{odds_of_losing_annotation_fn} \longleftarrow_{\Delta t=0} \text{hand_as_point_vals(hand)} : [0,1], \text{deck_holds(card, full_deck)} : [0.3, 1]$$

- `player_holds(C)`: When a card C is drawn, its corresponding point value v is used to update the lower bound of `player_holds(C)`. Each of the 52 cards has a dedicated rule.
- `hand_as_point_vals(hand)`: This rule updates the cumulative point value of the hand by appending each drawn card's value. It uses a non-grounded rule with an annotation function (`append_hand_annotation_fn`) to incrementally build the hand score. The threshold $[0.3, 1]$ ensures triggering for any card.
- `odds_of_losing(hand)`: Given the current hand value and remaining cards in the deck, this rule estimates the probability of exceeding 42 points on the next draw. The calculation is performed by the `odds_of_losing_annotation_fn`, using the fraction of risky remaining cards.

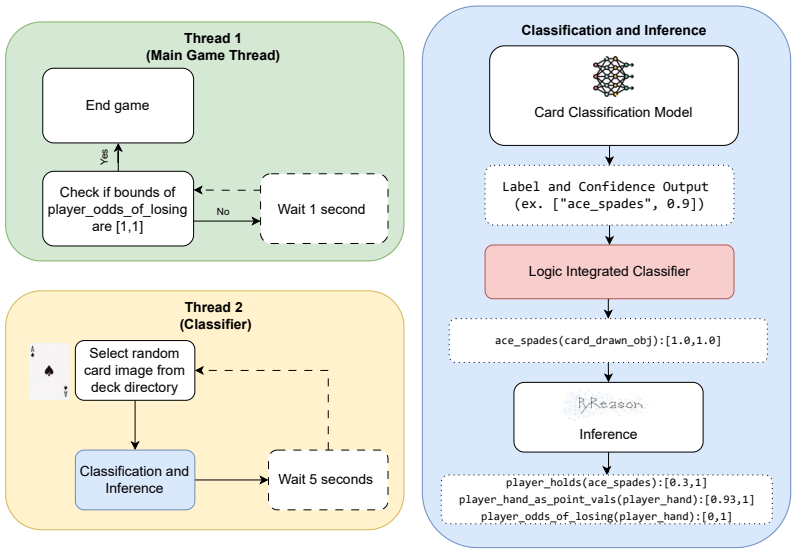


Figure 5: Workflow diagram for the card game simulation.

Figure 5 depicts the full simulation loop: one thread monitors termination conditions, while another periodically classifies drawn cards and updates the logical state. This concurrent design shows how PyReason's fixed-point inference can be driven by — and synchronized with — real (wall-clock) time. The classification and inference module periodically polls the image classifier to identify each drawn card and then encodes those results as PyReason facts, allowing for logical reasoning about the evolving game in the background.

4.2.2 Interpreting the Inference Trace

The explainable trace in Table 2 records every Fixed-Point Operation (FPO) in one run of the "42" game, showing how PyReason:

1. Asserts each drawn card via the `card_drawn_obj` fact at certainty $[1, 1]$.
2. Fires `player_holds` to add the card's point value v to the lower bound of `hand_as_point_vals`.
3. Fires `odds_of_losing` to compute the interval $[p, 1]$, where p is the fraction of remaining cards that would push the total above 42. When $[p, 1] = [1, 1]$, no safe draws remain and the game ends.

5 Conclusions

This paper presented a framework to bridge the gap between machine learning perception and logical reasoning by integrating ML models directly with the PyReason temporal logic engine. Our approach uses generalized annotated logic to convert real-valued model outputs into logical facts. It provides a standardized architecture via the `LogicIntegrationBase` and `TemporalLogicIntegratedClassifier` classes, creating a modular, real-time pipeline that requires no additional glue code.

The utility of the framework was demonstrated in two case studies: a welding scenario for real-time process control and a card game for managing uncertainty in decision making. These examples

Table 2: Simplified node traces for one simulation of the card game

FPO	Node	Label	Old Bound	New Bound
0	card_drawn_obj	two_clubs	[0.0,1.0]	[1.0,1.0]
1	two_clubs	player_holds	[0.0,1.0]	[0.6,1.0]
2	player_hand	hand_as_point_vals	[0.0,1.0]	[0.6,1.0]
3	player_hand	odds_of_losing	[0.0,1.0]	[0.0,1.0]
4	card_drawn_obj	ten_hearts	[0.0,1.0]	[1.0,1.0]
5	ten_hearts	player_holds	[0.0,1.0]	[0.6,1.0]
6	player_hand	hand_as_point_vals	[0.0,1.0]	[0.66,1.0]
7	player_hand	odds_of_losing	[0.0,1.0]	[0.0,1.0]
8	card_drawn_obj	six_clubs	[0.0,1.0]	[1.0,1.0]
9	six_clubs	player_holds	[0.0,1.0]	[0.6,1.0]
10	player_hand	hand_as_point_vals	[0.6,1.0]	[0.666,1.0]
11	player_hand	odds_of_losing	[0.0,1.0]	[0.0,1.0]
12	card_drawn_obj	four_clubs	[0.0,1.0]	[1.0,1.0]
13	four_clubs	player_holds	[0.0,1.0]	[0.6,1.0]
14	player_hand	hand_as_point_vals	[0.666,1.0]	[0.6666,1.0]
15	player_hand	odds_of_losing	[0.0,1.0]	[0.0,1.0]
16	card_drawn_obj	jack_spades	[0.0,1.0]	[1.0,1.0]
17	jack_spades	player_holds	[0.0,1.0]	[0.9,1.0]
18	player_hand	hand_as_point_vals	[0.6666,1.0]	[0.66669,1.0]
19	player_hand	odds_of_losing	[0.0,1.0]	[0.0,1.0]
20	card_drawn_obj	ace_clubs	[0.0,1.0]	[1.0,1.0]
21	ace_clubs	player_holds	[0.0,1.0]	[0.3,1.0]
22	player_hand	hand_as_point_vals	[0.66669,1.0]	[0.666693,1.0]
23	player_hand	odds_of_losing	[0.0,1.0]	[0.23913,1.0]
24	card_drawn_obj	three_hearts	[0.0,1.0]	[1.0,1.0]
25	three_hearts	player_holds	[0.0,1.0]	[0.6,1.0]
26	player_hand	hand_as_point_vals	[0.666693,1.0]	[0.6666936,1.0]
27	player_hand	odds_of_losing	[0.23913,1.0]	[1.0,1.0]

highlighted the system's ability to handle temporal dynamics and produce fully explainable inference traces, which are crucial for debugging and policy refinement.

Beyond the integration with machine learning models for perception, the PyReason framework is architecturally designed to support the integration of external, specialized reasoning engines, a concept analogous to the pluggable constraint solvers found in Constraint Logic Programming (CLP). This modularity allows the logic program to function as a high-level orchestrator that defines a problem space and coordinates with specialized computational modules. While the core of this paper focuses on integrating machine learning models for perception, the framework's underlying design is more general, enabling interfaces with various external solvers. A clear demonstration of this capability is presented in the work [11], which describes using PyReason to manage medical triage optimization. In their approach, the logic program dynamically formulates optimization problems by reasoning over situational facts to select the appropriate objectives and constraints. These problem instances are then passed to a dedicated external integer programming solver for resolution. Crucially, the solver's output is translated back into logical facts and reintegrated into the PyReason program, which then performs further sym-

bolic reasoning on these results to guide decision-making. This workflow exemplifies a general pattern where PyReason orchestrates a process by leveraging an external, "pluggable" engine for a specialized reasoning task, while maintaining a single, explainable logical framework.

Future work will focus on extending this integration in several key directions. In particular, we look to integrate "metacognitive" learned about the failure of the model to improve results on down-stream tasks (e.g., see [8]). In particular, early results have shown that the combination of such rules with abductive inference in PyReason can lead to improved ensembling of object detection ML models [9]. Based on those results, we look to examine further practical integrations for ML-logic programming integration with such error correction capabilities. Additionally, we plan to apply the framework to more complex, large-scale industrial scenarios and explore its use with a broader range of ML models, such as Large Language Models for business process automation, to further validate its robustness and versatility.

Acknowledgments

This research was supported by the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement No. HR00112420370 (MCAI). The views expressed in this paper are those of the authors and do not necessarily reflect the official policy or position of the U.S. Army, the U.S. Department of Defense, or the U.S. Government.

References

[1] Bowen Xi, Kevin Scaria, Divyagna Bavikadi & Paulo Shakarian (2025): *Rule-Based Error Detection and Correction to Operationalize Movement Trajectory Classification*. arXiv:2308.14250.

[2] Dyuman Aditya, Kaustuv Mukherji, Srikar Balasubramanian, Abhiraj Chaudhary & Paulo Shakarian (2023): *PyReason: Software for Open World Temporal Logic*. In: *AAAI Spring Symposium*, CEUR-WS.org. Available at `https://ceur-ws.org/Vol-3433/paper17.pdf`.

[3] Kareem Ahmed, Stefano Teso, Kai-Wei Chang, Guy Van den Broeck & Antonio Vergari (2022): *Semantic Probabilistic Layers for Neuro-Symbolic Learning*. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho & A. Oh, editors: *Advances in Neural Information Processing Systems*, 35, Curran Associates, Inc., pp. 29944–29959. Available at `https://proceedings.neurips.cc/paper_files/paper/2022/file/c182ec594f38926b7fcb827635b9a8f4-Paper-Conference.pdf`.

[4] Samy Badreddine, Artur d'Avila Garcez, Luciano Serafini & Michael Spranger (2022): *Logic Tensor Networks*. *Artificial Intelligence* 303, p. 103649, doi:10.1016/j.artint.2021.103649.

[5] Cristina Cornelio, Jan Stuehmer, Shell Xu Hu & Timothy Hospedales (2023): *Learning where and when to reason in neuro-symbolic inference*. In: *The Eleventh International Conference on Learning Representations*. Available at `https://openreview.net/forum?id=en9V5F8PR-`.

[6] Eleonora Giunchiglia, Alex Tatomir, Mihaela Cătălina Stoian & Thomas Lukasiewicz (2024): *CCN+: A neuro-symbolic framework for deep learning with requirements*. *International Journal of Approximate Reasoning* 171, p. 109124, doi:10.1016/j.ijar.2024.109124. Synergies between Machine Learning and Reasoning.

[7] Michael Kifer & V. S. Subrahmanian (1992): *Theory of Generalized Annotated Logic Programming and its Applications*. *J. Log. Program.* 12(3&4), pp. 335–367, doi:10.1016/0743-1066(92)90007-P.

[8] Joshua Shay Kricheli, Khoa Vo, Aniruddha Datta, Spencer Ozgur & Paulo Shakarian (2024): *Error Detection and Constraint Recovery in Hierarchical Multi-Label Classification without Prior Knowledge*. In: *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, CIKM '24, Association for Computing Machinery, New York, NY, USA, pp. 3842–3846, doi:10.1145/3627673.3679918.

[9] Mario Leiva, Noel Ngu, Joshua Shay Kricheli, Aditya Taparia, Ransalu Senanayake, Paulo Shakarian, Nathaniel Bastian, John Corcoran & Gerardo Simari (2025): *Consistency-based Abductive Reasoning over Perceptual Errors of Multiple Pre-trained Models in Novel Environments*. arXiv:2505.19361.

[10] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester & Luc De Raedt (2018): *DeepProbLog: neural probabilistic logic programming*. In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, Curran Associates Inc., Red Hook, NY, USA, p. 3753–3763. Available at `https://proceedings.neurips.cc/paper_files/paper/2018/file/dc5d637ed5e62c36ecb73b654b05ba2a-Paper.pdf`.

[11] Jaikrishna Manojkumar Patil, Adam Chapman, Richard Knuszka, John Chapman & Paulo Shakarian (2025): *Reasoning about Medical Triage Optimization with Logic Programming*. arXiv:2507.10781.

[12] Zhun Yang, Adam Ishay & Joohyung Lee (2020): *NeurASP: Embracing Neural Networks into Answer Set Programming*. In Christian Bessiere, editor: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, International Joint Conferences on Artificial Intelligence Organization, pp. 1755–1762, doi:10.24963/ijcai.2020/243. Main track.

VERUS-LM: a Versatile Framework for Combining LLMs with Symbolic Reasoning

Benjamin Callewaert Simon Vandevelde Joost Vennekens
Dept. of Computer Science
De Nayer Campus, KU Leuven, Belgium
Leuven.AI – KU Leuven Institute for AI
Flanders Make – DTAI-FET
{benjamin.callewaert, s.vandevelde}@kuleuven.be
Vrije Universiteit Brussel, Brussels, Belgium
joost.vennekens@vub.be

A recent approach to neurosymbolic reasoning is to explicitly combine the strengths of large language models (LLMs) and symbolic solvers to tackle complex reasoning tasks. However, current approaches face significant limitations, including poor generalizability due to task-specific prompts, inefficiencies caused by the lack of separation between knowledge and queries, and restricted inferential capabilities. These shortcomings hinder their scalability and applicability across diverse domains. In this paper, we introduce VERUS-LM, a novel framework designed to address these challenges. VERUS-LM employs a generic prompting mechanism, clearly separates domain knowledge from queries, and supports a wide range of different logical reasoning tasks. This framework enhances adaptability, reduces computational cost, and allows for richer forms of reasoning, such as optimization and constraint satisfaction. We show that our approach succeeds in diverse reasoning on a novel dataset, markedly outperforming LLMs. Additionally, our system achieves competitive results on common reasoning benchmarks when compared to similar state-of-the-art approaches, and significantly surpasses them on the difficult AR-LSAT dataset. By pushing the boundaries of hybrid reasoning, VERUS-LM represents a significant step towards more versatile neurosymbolic AI systems. All code and datasets required to reproduce the results in this text are available online: https://gitlab.com/EAVISE/bca/verus-lm

1 Introduction

Logical reasoning is an essential aspect of problem-solving, decision-making, and critical thinking, making it an important goal in Artificial Intelligence. While recent large language models (LLMs) such as GPT-4 [2] and Gemini [25] have shown an impressive leap in reasoning-like capabilities through techniques like chain-of-thought prompting, they often lack the ability to guarantee faithful and transparent reasoning. Indeed, at a fundamental level, LLMs operate as black-box probabilistic models, which makes it difficulty to ensure accurate and coherent reasoning steps. As a consequence, whether they are capable of "true reasoning" is still subject of much debate [4, 30, 22].

Symbolic inference engines, on the other hand, can derive conclusions that are provably correct and typically also explainable. However, these systems struggle to interpret ambiguous natural language input. This creates an opportunity for neurosymbolic methods to combine the strengths of LLMs with symbolic reasoning systems. Interestingly, this method loosely parallels Kahneman's theory on human cognition, in which reasoning is divided into fast and unconscious thinking (System 1) and slow and methodical thinking (System 2) [15]. Such approaches have already demonstrated impressive improvements on common reasoning datasets [18, 14, 17].

Despite these promising results, current state-of-the-art approaches have several limitations that hinder their applicability across diverse domains. One significant drawback is their heavy reliance on

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 47–62, doi:10.4204/EPTCS.439.5

© B. Callewaert, S. Vandevelde & J. Vennekens
This work is licensed under the
Creative Commons Attribution License.

task-specific prompts to generate symbolic representations, via in-context learning. This makes these approaches less general and harder to adapt to new domains, since this may require manual prompt engineering. In real-world scenarios, where problems are highly variable, task-specific prompts can cause a severe bottleneck. Furthermore, they typically try to reduce every user query to the same logical reasoning task (or to one of only a very limited set of reasoning tasks). In many domains, users have widely different kinds of queries, which correspond naturally to equally wide range of logical reasoning tasks (e.g., optimization, generating solutions, deriving consequences, and generating explanations). Therefore, there is a lack of *versatility* in the types of queries they are able to answer, limiting their effectiveness in complex domains where multiple forms of reasoning are required.

In this paper, we propose an enhanced neurosymbolic framework that addresses these limitations titles VERUS-LM. It features a generic prompting pipeline, a semantic refinement step, a clear separation between declarative knowledge from questions, and support for a wide range of logical reasoning tasks. In this way, it is able to handle complex and dynamic environments in a robust and effective manner.

2 Related Work

Improving the reasoning capabilities of LLMs has recently become a topic of great interest. For the purpose of this paper, however, we will not discuss works on improving LLM reasoning with in-context prompting or by fine-tuning, but refer to [13, 19] for more information on those techniques. Instead, we will focus on the neurosymbolic approach in which an LLM is coupled with an explicit symbolic reasoning engine.

Current state-of-the-art systems all largely work in the same way. Given a reasoning problem, which typically consists of a description of some domain and a query that needs to be answered, they first let an LLM generate a formal representation of the problem and then pass this on to a reasoning engine to solve. In this way, they combine the benefits of both kinds of AI systems: the LLM offers a user-friendly natural language interface, while the reasoning engine ensures correct reasoning. This offers a promising path towards improving the reasoning skills of LLM-based systems. We will now briefly go over some of these systems and touch on how they differ.

In [14], the authors present a pipeline to automatically generate Answer Set Programming (ASP) representations [5]. Their method uses the LLM three times: first to extract relevant constants, next to generate predicates, and finally to generate the ASP rules. Using a logic puzzle dataset, they demonstrate a 71% increase in accuracy compared to a baseline GPT-4. The same authors also rely on ASP in the [LLM]+ASP framework [32], where given a problem description, the LLM extracts a set of atomic facts. These facts, coupled with pre-defined, hand-crafted "knowledge modules" containing domain-specific ASP rules, are fed to an ASP solver to find a solution. They validate their approach on four "story-based" datasets, on which it outperforms the state-of-the-art.

In the LINC system [17], an LLM is used to generate statements in First Order Logic (FOL), which are given to a FOL reasoner. To mitigate formalization errors, they generate K formalizations and make use of K-way majority voting to decide on the correct response. They demonstrate that their approach is competitive on the FOLIO dataset [12], and achieves remarkable improvements on the ProofWriter dataset [24].

Finally, though similar to the previous approaches, Logic-LM [18] has two distinctive features. First, instead of relying on a single formalization language, their system supports logic programs, FOL, constraint satisfaction problems and SAT encodings. In this way, the system is able to handle a broader range of problem types. Second, they introduce a *self-refinement stage*, in which the LLM iteratively attempts

to fix any syntax errors based on feedback from the solver. They validate their approach on five datasets, on which they generally outperform GPT-4 using chain-of-thought as baseline.

It is worth noting that most of the aforementioned methods rely heavily on dataset-specific examples, utilized for in-context learning, to guide the LLMs. Furthermore, they only support a limited number of logical reasoning tasks, which limits their generalization capabilities to new datasets. This is especially apparent on the AR-LSAT dataset [33], featuring logical reasoning questions from the Law School Admission Test, which is high in diversity and thus requires broad reasoning capabilities. For instance, given knowledge on the types of CDs sold by a store, the questions can range from "Which CDs are on sale?" to "Given ϕ, which statements are true?" and "What are the minimum CDs required to be on sale if ϕ?", for some additional statement ϕ. Though it still improves on the baseline GPT-4, Logic-LM only achieves a 43.04% accurary on this dataset due to its diversity.

Though different from the others, we also briefly discuss the SymbCoT framework [31] for the sake of comparison. Like the aforementioned works, SymbCoT internally translates a problem into FOL using in-context learning. However, instead of using a separate reasoning engine, the LLM itself then performs logical reasoning on the generated statements in a step-by-step way similar to Chain-of-Thought, instructed by specific prompts that try to mimic the kinds of logical reasoning found in logical solvers. By not using a separate reasoning engine, the approach becomes more robust to syntax errors. Interestingly, this approach consistently outperforms standard Chain-of-Thought and Logic-LM on the PrOntoQA [21], FOLIO and ProofWriter datasets.

3 System Design

3.1 Requirements

As discussed in the previous section, current state-of-the-art approaches typically focus on a single reasoning tasks. However, more realistic use cases will likely require a more versatile approach. Imagine, for instance, a chat bot acting as a digital assistant in the field of car insurance. A person interacting with such an AI system will not only ask questions like "Am I eligible for an insurance policy?", but will also ask other things such as "Why am I (in)eligible?", "Depending on my car type, what would my insurance premium?" and "What elements can I change to minimize my premium?".

For a system to be capable of handling such use cases, there are two main requirements. First, the system must support multiple modes of reasoning, which means that reasoning-specific LLM prompts will not work well (as evidenced by the results of state-of-the-art systems on the AR-LSAT dataset). Second, the reasoning component of such a system must be capable of performing different kinds of reasoning, either using a single versatile engine or, similar to Logic-LM, using multiple ones. However, this latter approach has the downside that the domain knowledge needs to be formalized anew for each type of reasoning, increasing the computational cost and risk of errors. We therefore prefer the former approach of using a single reasoning engine that supports different forms of reasoning.

Given this, we have the following design requirements for VERUS-LM:

1. *Versatile*: it should support multiple *useful* forms of reasoning (e.g., verification, explanation, optimisation, etc.)
2. *Knowledge reuse*: it should distinguish between "domain knowledge" and "task to be performed", allowing the domain knowledge to be reused for different tasks.
3. *Generic*: all aspects of the tool, including the prompting method, should be independent of the type of reasoning tasks or the kind of domain that is being considered.

3.2 Reasoning Engine

While the aforementioned requirements pertain to the general design of our system, they also constrain our selection of reasoning engine. To serve as the main reasoning core of VERUS-LM, we have selected the IDP-Z3 system [6]. IDP-Z3 is a reasoning engine for FO(·), a rich extension of classical First-Order Logic with useful features such as types, aggregates, (inductive) definitions, and more. It explicitly supports the Knowledge Base Paradigm [8], in which the same domain knowledge can be used for many different reasoning tasks.

We will briefly illustrate FO(·) and the forms of reasoning supported by IDP-Z3 through a small example, referring to [6] for a more extensive overview.

As in normal FOL, a *vocabulary* in FO(·) consists of a set of symbols. Because FO(·) is a typed logic, this includes types, in addition to predicate and function symbols. For instance, in the below example about car insurance, we use types `Customer` and `Car`. The function `risk_factor` maps each `Car` to its associated risk factor ($\in \mathbb{R}$), while the function `age` maps each `Applicant` to their age ($\in \mathbb{Z}$). A number of constants (= 0-ary functions) represent the car's value, its type, and the insurance premium. Finally, a unary predicate `applicant` represents which of the customers is requesting the insurance. Again following FOL, a *structure* for a vocabulary provides an interpretation for some (not necessarily all) of the symbols in this vocabulary. Finally, a *theory* consists of a set of logical sentences. In this simple example, we have two sentences: one states that every applicant must be an adult (L18), and one defines the calculation of the insurance premium (L19).

```
1  vocabulary V {
2   type Customer := {Ann, Brit}
3   type Car := {Sedan,SUV,...}
4   risk_factor: Car → Real
5   age: Applicant → Int
6   car_value: → Real
7   premium:  → Real
8   car_type:  → Car
9   premium: → Real
10  applicant: Customer → Bool
11 }
12
13 structure S:V {
14  age := {Ann → 16, Brit → 32}.
15  risk_factor := {Sedan→1.03,Truck→1.15, ...}.
16 }
17
18 theory T:V {
19  ∀p in Customer: applicant(p) ⇒ age(p) ≥ 18.
20  premium() = (car_value()/100) × risk_factor(car_type()).
21 }
```

As usual, we denote the value of a symbol σ in a structure S by σ^S and extend this notation to terms and formulas (e.g., `age(Ann)`$^S = 16$). Also as usual, a structure S such that $\phi^S = \texttt{true}$ for all $\phi \in T$ is called a *(logical) model* of T and this is written as $S \models \phi$.

Given such a KB, IDP-Z3 can perform different forms of reasoning to answers different questions.

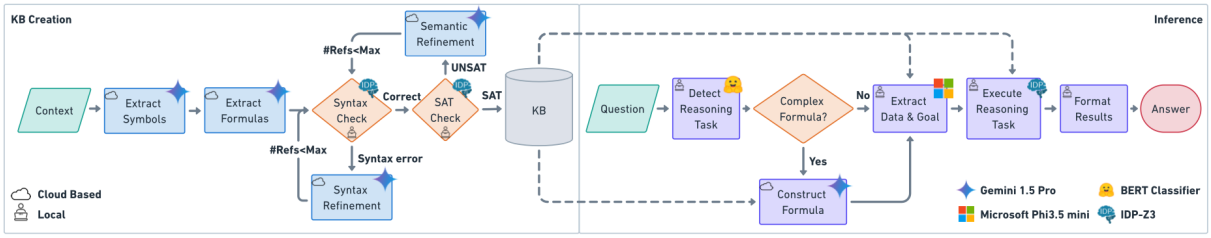


Figure 1: Flowchart depicting steps of VERUS-LM framework and the used system in each step

For instance, IDP-Z3 can determine which customers are eligible for an insurance policy, and explain why. Given the car type and car value, it can also calculate the insurance premium. Or, it can "optimize" the premium by looking for the car type with the lowest risk factor. Using these different reasoning tasks does not require any modifications to the KB itself, allowing for straightforward knowledge reuse.

These properties of IDP-Z3 make it a natural choice for our reasoning system. Additionally, FO(·) is expressive enough to model many problem domains, as supported by real-life use cases [3, 9, 26, 27]. Furthermore, as First-Order Logic is the subject of many articles and textbooks, the training corpora of LLMs will also contain enough information to allow them to be fluent in this formalism.

3.3 VERUS-LM Architecture

The VERUS-LM framework integrates natural language processing and formal reasoning through a structured, two-phase process, as depicted in Figure 1. The first phase, Knowledge Base Creation, uses an LLM to translate domain knowledge into a symbolic FO(·) specification. Note that, in this phase, we do not yet consider specific questions and instead create a reusable KB. Questions are only considered in the second phase, the Inference phase, where they are interpreted and then answered by means of an appropriate call to one of IDP-Z3's reasoning tasks. In other words, when given multiple questions about the same domain it is only formalized once, greatly reducing the computational cost.

4 Knowledge Base Creation

The first phase of the VERUS-LM framework consists of transforming domain knowledge into a formal KB. This translation is performed in three steps, which we will discuss in the following sections.

4.1 Symbol Extraction

The LLM first identifies the relevant types, functions, and predicate symbols for the problem domain. We use a prompt that describes these three different kinds of symbols and how they are typically used. Important here is that we do not include examples from a specific dataset. Instead, our prompt just provides general instructions on how to identify and categorize the different concepts, and provides three general examples that do not appear in any of the datasets as an illustration. The prompt also contains the instruction to annotate each symbol with its informal language meaning, so that this information is present in the next phases of the pipeline.

One limitation of IDP-Z3 is that it does not support reasoning under the Open World Assumption, which is necessary for the FOLIO and Proofwriter datasets. Instead, IDP-Z3 always makes the Closed

Word Assumption (CWA). In particular, it requires that the extension of all types (apart from built-in types $\mathbb{R}$ and $\mathbb{Z}$) is always fully enumerated. As a work-around, VERUS-LM simulates OWA reasoning by introducing an additional "unknown" domain element into each type. Even though this is not theoretically correct, it persistently yields the correct results in practice, as shown in the experiments described further on.

4.2 Formula Extraction

During the formula extraction step, an LLM generates an FO($\cdot$) theory that corresponds to the description of the problem domain. To guide this generation process, a brief overview of the FO($\cdot$) grammar and clear instructions are presented, supported by two illustrative examples. These examples adopt a step-by-step approach, pairing each logical sentence with its corresponding natural language meaning for clarity. Additionally, the LLM is instructed to also consider implicit and commonsense knowledge, demonstrated in the examples. At the end of the prompt we add the previously generated vocabulary, along with the natural language description of the problem to formalize.

4.3 Self-Refinement

Although in our experience the LLM often produces correct logical sentences, errors still occur. To correct these, we introduce a self-refinement step that uses feedback from the reasoning engine to correct erroneous logical statements. This approach is inspired by recent work on imperative programming [7, 16] and the Logic-LM framework [18]. However, whereas these approaches only consider syntactic correctness, we introduce a novel second step to take semantic correctness into account as well, by means of a satisfiability check. These refinements are part of an iterative process, where VERUS-LM will repeat them until no errors are found or a maximum of attempts is reached.

Syntax Refinement If the reasoning engine finds a syntax error, the LLM is instructed to correct its output, i.e., either the vocabulary and/or the theory it produces. To this end, the LLM is prompted with its previous erroneous output, the solver's error message, and some examples of common syntax errors and their remedies. Detailed error descriptions are provided by FOLINT [28], IDP-Z3's code analysis tool.

Semantic Refinement Once the KB is syntactically correct, we use a satisfiability check to find additional semantic errors. It is reasonable to assume that the KB should be satisfiable, because, at this point, it contains only a formalization of the provided domain knowledge. For instance, if we were solving a planning problem, the KB would contain a description of the actions that can be executed, but we would not yet include a specific goal that the plan should achieve. If this KB is already inconsistent, then we take this as a sign that the LLM made a mistake in translating the domain knowledge and we instruct it to refine its previous output, based on an explanation of why this was unsatisfiable. IDP-Z3 can give such an explanation itself, by generating a "minimal unsatisfiable subset" of conflicting assignments and constraints [6].

5 Inference

Once the KB is constructed, the inference phase attempts to correctly answer the question. To this end, VERUS-LM follows a series of steps, as illustrated on right side of Figure 1. Given a question,

the framework starts by identifying the intended reasoning task and extracting the relevant information, which are passed on to the reasoning engine. Once the latter has finished, its output is formatted back to natural language.

5.1 Detect Reasoning Task

First, VERUS-LM classifies the user's question into one of the following eight forms of reasoning.

1. **Model Generation/Expansion**: generate n logical models of the theory T (i.e., structures S such that $S \models T$). Possibly, the interpretation σ^S that some of the symbols σ should have in this model S is already given (e.g., we might provide the value for constants `car_value` and `car_type` and ask the reasoner to complete the interpretation for the remaining symbols).
2. **Satisfiability**: verify if at least one model S exists for a given theory T.
3. **Optimization**: find the model S of a given T in which a given term t reaches its minimal/maximal value t^S.
4. **Propagation**: determine which atomic formulas are true / false in *all* models S of the given theory T.
5. **Explain**: explain why a given atomic formula is true / false in all models of T, or, in T has no models, then explain the inconsistency.
6. **Determine Range**: determine the range of possible values for a given function term $f(\vec{t})$ given a theory T, i.e., the set of all values v such that there exists at least one model S of T in which $f^S(\vec{t}^S) = v$.
7. **Relevance**: determine which symbols σ are relevant, in the sense that there exists a model S of T such that S would no longer be a model if the value of σ in S were different.
8. **Logical entailment**: verify whether a given statement ϕ is logically entailed by theory T.

All of these were either already directly offered by IDP-Z3 (1-7) or trivial to implement (8). This demonstrates that IDP-Z3 indeed provides the versatility that is required by the VERUS-LM framework.

Detecting the correct inference to be executed is crucial to ensure that questions are correctly answered. To detect the intended inference from the a question, we fine-tuned a BERT sentence classifier [20] on a custom dataset, which contains over 1000 natural language questions and their corresponding logical reasoning task.

To address more complex nested questions that demand a sequence of logical reasoning steps, we developed a multi-step reasoning mode. Similar to Chain-of-Thought prompting, this mode first breaks the complex question into a series of steps. Each step is then treated as an independent reasoning task, after which the results are combined.

5.2 Information Extraction

Several of the forms of reasoning listed above require some additional information, in addition to a theory T. We distinguish two cases.

Extracting data and goal detection Typically, in addition to a theory T, an interpretation is given for at least some of the symbols of T. For instance, the problem statement may be: "Given that Ann is 16 and Brit is 32, who is eligible for insurance?" Such information should be extracted and included as part of a structure on which the reasoning is performed (in this case, $age^S = \{\text{Ann} \mapsto 16, \text{Brit} \mapsto 32\}$). In addition, some other reasoning tasks also require additional information of similar complexity, e.g., optimization requires that we know which term t should be minimized / maximized.

This information extraction task is made significantly easier by the fact that, at this point, we already know the vocabulary. We can therefore use language model grammars, such as llama.cpp GNBF grammar [10], to force the output to adhere to an automatically generated KB-specific grammar, ensuring that only correctly typed interpretations are generated. As we will show, this pruning of the output space allows us to achieves the same performance for this task with a small language model (SLM) as with a larger LLM. Since such a SLM can run on a standard laptop, this may significantly reduce computational costs, as well as enhance data security, by avoiding the transfer of potentially sensitive information to external servers.

Construct complex formulas In some case, such as for the logical entailment reasoning task, more complex additional information is required at inference time (e.g., the formula ϕ for which it should be checked whether $T \models \phi$). Constructing a complex formula ϕ from the natural language question is of similar complexity as constructing the knowledge base in the first phase. Therefore, SLMs are ill-suited for this task, and a larger LLM should be used instead. Similarly to the KB creation step, this LLM not only generates formulas but may also add new symbols to the vocabulary.

We currently do not attempt to automatically detect whether a specific question contains complex formulas or not, and thus requires an LLM, because we may wish to make this decision depending on the application. If we do not care about computational cost, we can just always invoke the LLM for optimal accuracy. Alternatively, we could base the decision whether to use an SLM or LLM purely on the detected reasoning task (e.g., Entailment is typically more likely to need an LLM), or we could even train a specific classifier to make the decision.

6 Experiments

6.1 Validation

We first conduct a basic validation of VERUS-LM by checking (1) whether our use of a reasoning engine indeed leads to better perfomance on logical reasoning tasks than a stand-alone LLMs and (2) that the VERUS-LM pipeline is indeed able to correctly identify our eight different reasoning tasks from natural language text. Since existing datasets typically cover only one or at most a few reasoning tasks, we constructed our own dataset called *DivLR* in which all eight reasoning tasks are present. It consists of 115 questions covering six domains: investment **S**trategies, **C**OVID restrictions, water **I**rrigation needs, a **H**andyman and two variants in the Body Mass Index (BMI) domain. The reason for having two BMI variants is that information about BMI (i.e., how to calculate it, the different risk levels associated to BMI values, etc.) probably occurs in LLM training data. In addition to our **B**MI domain which contains the well-known definition, we also defined a **B*** domain in which novel formulas and ranges for BMI were made up, similar to the experiment in [11].

We compare VERUS-LM to baseline language models on this dataset. We compare with both an SLM and an LLM, and consider two versions of our pipeline: one in which our information extraction

Table 1: Results of VERUS-LM with SLM and LLM for information extraction and SLM and LLM separately as baseline. SLM = Phi 3.5 mini instruct, LLM = Gemini 1.5 Pro

System	C	S	H	I	B	B*	Avg
V-SLM	**94.7**	86.4	86.7	86.4	61.1	66.7	80.3
V-LLM	89.5	**100**	**93.3**	**95.5**	**88.9**	**83.3**	**91.8**
SLM	47.4	50	53.3	36.4	38.9	11.1	39.5
LLM	73.7	86.4	60	63.6	66.7	50	66.7

Table 2: Distribution of different reasoning tasks in custom datasets, detection accuracy of the BERT classifier and execution accuracy of VERUS-LM with an SLM (V-SLM) and LLM (V-LLM) for information extraction.

Reasoning Task	%	Detect Acc	Exec_Acc	
			V-SLM	V-LLM
Model Expansion	1.8	100	100	100
Satisfiability	9.6	81.8	100	90.9
Optimization	26.3	90.0	93.3	96.7
Propagation	28.1	76.5	68.6	91.2
Explain	8.8	100	80.0	90.0
Determine Range	14.0	92.9	100	85.7
Relevance	1.8	100	100	100
Entailment	9.6	100	27.3	90.9

is done by an SLM (V-SLM) and one in which this is done by an LLM (V-SLM). the LLM is Gemini 1.5 Pro [25] and the SLM is a quantised version of Phi 3.5 [1], a 3.8-billion-parameter language model whose inference cost is about 10,000 times lower than large models like GPT-4. All experiments were performed on a Mac M2 Pro CPU with 32GB of RAM.

Table 1 shows that both V-SLM and V-LLM consistently outperform both the SLM and the LLM on their own. However, V-SLM performs significantly worse on the two BMI domains than on the other domain, and also significantly worse than V-LLM on these domains. A deeper analysis shows that the SLM typically refuses to simply extract the information that the reasoning engine needs, and instead tries to perform BMI-calculations itself. We conjecture that this is due to the BMI-related examples that are likely part of the data on which the SLM was trained.

A similar but less striking issue can be seen when comparing the results of the LLM on both the **B** and **B*** domain: the observed drop in accuracy (66.7→50) can be explained by the fact that the LLM has memorised the correct definition of BMI from its training data, and that is unable reason with a different definition, when one is explicitly provided. This suggests that, more generally, if an LLM at some point learns out-dated facts, it is hard to correct these in the prompt, requiring retraining the network instead. We see that same issue is still somewhat observable in V-LLM (88.9→83.3), but here the reasoning engine pipeline substantially reduces the effect.

Table 2 confirms that our BERT classifier identifies the correct logical reasoning task with an average accuracy of 92.6%, performing consistently well on all tasks (with a small dip for Propagation). For the subsequent information extraction, we see that, for most of the reasoning tasks, V-LLM and S-SLM

Table 3: Overview of datasets used for comparison with state-of-the-art

Dataset	Example	For inference	# Options	Reasoning	#
PrOntoQA [21]	"Stella is not hot"	Data	2: $\top$, $\bot$	Satisfiability	500
ProofWriter [24]	"The rabbit visits the cat"	Data	3: $\top$,$\bot$, ?	Propagation	600
FOLIO [12]	"Marvin is neither a human nor from Mars"	Formulas	3: $\top$, $\bot$, ?	Logical entailment	204
LogicalDeduction [23]	A)"Eve finished third.", B) "Max finished third", ...	Only Data	3-5-7	Satisfiability	300
AR-LSAT [33]	"What is the max #CDs on sale? A) two, B) six, ..."	Formulas	5	Multiple	231

Table 4: Comparison of the results of the neurosymbolic system. *Results from original paper. †Results from [31]. The results for Logic-LM are those with self refinements included, but without the LLM fallback in case of recurring syntax error.

System	ProntoQA	Proofwriter	Folio	Logical Deduction	AR-LSAT	Average	ST Dev
VERUS-LM	95.8	*93.83*	*78.43*	*88.67*	**68.36**	**85.02**	**11.49**
Logic-LM*	83.20	78.80	68.55	87.63	23.40	68.32	26.09
LINC*	x	**98**	72.50	x	x	x	x
SymbCoT*	**99.60**	82.50	**83.33**	**93.00**	*43.91*	*80.47*	21.63
GPT4†	77.40	52.67	69.11	71.33	33.33	60.77	*17.86*
GPT4 - CoT†	*98.79*	68.11	70.58	75.25	35.06	69.56	22.80

perform about equally well. The biggest difference is seen for Entailement. This is the only reasoning task in our DivLR dataset for which complex formulas need to be constructed at inference time, which V-SLM predictably struggles with (accuracy 29.6%). For the Propagation task, we also observe a difference in accuracy (-25.6%) between V-SLM and V-LLM. This is primarily due V-SLM struggling with the BMI domain, as already discussed.

Overall, we conclude that the proposed pipeline makes sense: (1) introducing a logical reasoning engine into the pipeline indeed leads to better performance than using only a language model; (2) different logical reasoning tasks can indeed by successfully identified by a simple BERT classifier; (3) all of the necessary information for the reasoning tasks can indeed be extracted with a small language model, as long as no complex formulas need to be constructed, and with the caveat that an SLM might be less robust when handling concepts that already occurred frequently in its training data.

In our following experiments, we have always used V-SLM for benchmark datasets that do not require the construction of complex formulas at inference time and V-LLM otherwise.

Table 5: The distribution of reasoning tasks detected in AR-LSAT and VERUS-LM's accuracy

Reasoning task	Distribution (%)	Exe_Acc (%)
Optimization	6.06	71.43
Determine range	16.88	79.49
Entailment	45.89	66.04
Satisfiability	31.17	69.01

6.2 Comparison to state-of-the-art

Having established that the VERUS-LM pipeline makes sense, we now compare it to the state-of-the-art on a number of standard benchmarks, summarized in Table 3. PrOntoQA, ProofWriter, FOLIO, and LogicalDeduction each target a specific reasoning task, while the challenging AR-LSAT dataset spans a broader range of reasoning tasks.

We compare VERUS-LM against three LLM-based approaches and two neurosymbolic approaches for logical reasoning: (1) GPT-4 answering directly, (2) GPT-4 with Chain-of-Thought, (3) Linc [17] (only FOLIO and ProofWriter), (4) Logic-LM [18] and (5) SymbCot [31].

As shown in Table 4, all systems achieve a roughly similar performance on the first four benchmarks, with VERUS-LM scoring at most 5% worse than the best system on each, without relying on **dataset-specific prompts**. For the challenging AR-LSAT benchmark, VERUS-LM significantly outperforms all other methods, doing about **25% better** than the second best system. We believe that this is due to (1) the **generality** of our approach, which does not use task-specific examples for in-context learning; (2) **FO(·)'s expressiveness** which allows for, e.g., straightforward representations of aggregates, such as "At least three CDs are on sale": $\#\{c \in CD : on_sale(c)\} \geq 3$; (3) our support for **different forms or reasoning**, four of which were detected by the classifier and executed, as shown in Table 5.

6.3 Effect of Self-Refinement

Table 6 presents the results for each dataset of: (1) VERUS-LM without performing any refinements after the initial KB creation; (2) VERUS-LM with only syntactic refinement; (3) complete VERUS-LM with also semantic refinements. In each case, we report the *execution rate*, which is the percentage of cases in which the knowledge base was syntactically correct and satisfiable. We also report the *execution accuracy*, which is the percentage of the those for which the reasoning engine returned a correct result. We also report the product of the two as "total accuracy".

The syntactic refinement increases the execution rate by 11.2% on average. The execution accuracy stays roughly the same, showing that KBs that had to be syntactically corrected are about as likely to correctly represent the domain knowledge as KBs that were initially already syntactically correct. When we then also include our semantic refinement, there is an additional increase in execution rate of on average 10%. This concerns cases in which the original KB was unsatisfiable, e.g., because of contradictory formulas or mistakes in the typing of a function or constant. Such errors typically arise only when the natural language description is difficult to understand, which also makes them hard to rectify in a correct way, as evidenced by the drop in execution accuracy when adding the semantic refinement. However, the total accuracy still markedly increases, showing that the semantic refinement step is indeed useful.

Table 6: Execution Rate and Accuracy of VERUS-LM in different refinements scenario's: **No** Refinements, with **Syntax** Refinements and with **Both** Syntax and Semantic Refinements

	Refs	Exe_Rate	Exe_Acc	Total_Acc
PrOntoQA	No	74	97	71.8
	Syntax	88.2 ↑ 14.2	97.3 ↑ 0.3	86.8 ↑ 15
	Both	98.2 ↑ 10	97.6 ↑ 0.3	95.8 ↑ 9
ProofWriter	No	90	95.7	86.2
	Syntax	92.3 ↑ 2.3	95.7 =	88.3 ↑ 2.1
	Both	99 ↑ 6.7	94.8 ↓ 0.9	93.8 ↑ 5.5
FOLIO	No	71.6	80.8	57.8
	Syntax	89.2 ↑ 17.6	80.6 ↓ 0.2	74 ↑ 16.2
	Both	100 ↑ 10.8	78.4 ↓ 2.2	78.4 ↑ 4.4
Logical Deduction	No	93.3	89.6	83.6
	Syntax	93.7 ↑ 0.4	89.7 ↑ 0.1	84 ↑ 0.4
	Both	99.3 ↑ 6.3	89.3 ↓ 0.4	88.7 ↑ 4.7
AR-LSAT	No	60.2	84.3	50.8
	Syntax	81.8 ↑ 21.6	78.2 ↓ 6.2	64 ↑ 13.2
	Both	98.7 ↑ 16.9	69.3 ↓ 8.9	68.4 ↑ 4.4

7 Limitations

Though VERUS-LM expands on the state of the art, it faces some limitations. Like the earlier papers in literature, VERUS-LM itself does not have a method for validating formula correctness. This is an inherent problem in this type of neuro-symbolic AI in general, as natural language can be ambiguous, and LLMs can always produce errors. To address this limitation, we are looking into bringing a domain expert in the loop without prior logic experience to validate the output [29].

Additionally, the expressiveness of FO($\cdot$) has limitations, making it unsuitable for, e.g., representing higher-order logic or for reasoning over OWA. Though VERUS-LM's simulated OWA performed well on the datasets discussed in this work, it remains an approximation and thus will not be effective for all situations. Similarly, our pipeline's semantic refinement step is also not always applicable, as there could be situations (such as reasoning over fake news) where we *can* expect unsatisfiability. Due to VERUS-LM's relience on a logical reasoning engine, it also inherits some of its limitations, such as SAT problems in very large domains. Furthermore, though the KB-creation of VERUS-LM is based on a generic prompting pipeline, its accuracy in entirely different logical paradigms remains a uncertain.

8 Conclusion

This paper introduced VERUS-LM, a versatile neurosymbolic framework that integrates language models with symbolic reasoning capabilities. Its two-phased approach separates Knowledge Base creation from a separate inference phase, in which multiple different reasoning tasks can be performed on the same knowledge base. This approach has the inherent advantage that multiple questions about the same domain can be answered in a computationally efficient way, by reusing the same knowledge base. Another improvement made by VERUS-LM is that we extend the state-of-the-art syntactic refinement step with a semantic refinement step, based on a satisfiability check. Our experimental analysis shows that this

indeed improves results. VERUS-LM significantly outperforms the state-of-the-art on the challenging and diverse AR-LSAT dataset, wile remaining competitive on simpler benchmarks.

References

[1] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl et al. (2024): *Phi-3 technical report: A highly capable language model locally on your phone*. arXiv preprint arXiv:2404.14219, doi:10.48550/arXiv.2404.14219.

[2] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat et al. (2023): *Gpt-4 technical report*. arXiv preprint arXiv:2303.08774, doi:10.48550/arXiv.2303.08774.

[3] Bram Aerts, Marjolein Deryck & Joost Vennekens (2022): *Knowledge-based decision support for machine component design: A case study*. Expert Systems with Applications 187, p. 115869, doi:10.1016/j.eswa.2021.115869.

[4] Emily M. Bender, Timnit Gebru, Angelina McMillan-Major & Shmargaret Shmitchell (2021): *On the Dangers of Stochastic Parrots: Can Language Models Be Too Big?* In: Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, FAccT '21, Association for Computing Machinery, New York, NY, USA, pp. 610–623, doi:10.1145/3442188.3445922.

[5] Gerhard Brewka, Thomas Eiter & Mirosław Truszczyński (2011): *Answer set programming at a glance*. Communications of the ACM 54(12), p. 92–103, doi:10.1145/2043174.2043195.

[6] Pierre Carbonnelle, Simon Vandevelde, Joost Vennekens & Marc Denecker (2023): *Interactive configurator with FO(.) and IDP-Z3*. arXiv preprint arXiv:2202.00343, doi:10.48550/arXiv.2202.00343.

[7] Xinyun Chen, Maxwell Lin, Nathanael Schärli & Denny Zhou (2023): *Teaching large language models to self-debug*. arXiv preprint arXiv:2304.05128, doi:10.48550/arXiv.2304.05128.

[8] Marc Denecker & Joost Vennekens (2008): *Building a Knowledge Base System for an Integration of Logic Programming and Classical Logic*. In Maria Garcia de la Banda & Enrico Pontelli, editors: Logic Programming, 5366, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 71–76, doi:10.1007/978-3-540-89982-2_12. Available at `http://link.springer.com/10.1007/978-3-540-89982-2_12`. Series Title: Lecture Notes in Computer Science.

[9] Marjolein Deryck, Jo Devriendt, Simon Marynissen & Joost Vennekens (2019): *Legislation in the Knowledge Base Paradigm: Interactive Decision Enactment for Registration Duties*. In: Proceedings of the 13th IEEE Conference on Semantic Computing, IEEE, pp. 174–177, doi:10.1109/icosc.2019.8665543.

[10] G. Gerganov (2023): *llama.cpp: LLM inference in C/C++*. Available at `https://github.com/ggerganov/llama.cpp`. Accessed: May 30, 2023.

[11] Alexandre Goossens, Simon Vandevelde, Jan Vanthienen & Joost Vennekens (2023): *GPT-3 for Decision Logic Modeling*. In: Proceedings of the 17th International Rule Challenge and 7th Doctoral Consortium @ RuleML+RR 2023 Co-Located with 19th Reasoning Web Summer School (RW 2023) and 15th DecisionCAMP 2023 as Part of Declarative AI 2023, CEUR Workshop Proceedings. Available at `https://ceur-ws.org/Vol-3485/`.

[12] Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson et al. (2022): *Folio: Natural language reasoning with first-order logic*. arXiv preprint arXiv:2209.00840, doi:10.48550/arXiv.2209.00840.

[13] Jie Huang & Kevin Chen-Chuan Chang (2023): *Towards Reasoning in Large Language Models: A Survey*. In: Findings of the Association for Computational Linguistics: ACL 2023, Association for Computational Linguistics, doi:10.18653/v1/2023.findings-acl.67.

[14] Adam Ishay, Zhun Yang & Joohyung Lee (2023): *Leveraging Large Language Models to Generate Answer Set Programs*. In: Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, pp. 374–383, doi:10.24963/kr.2023/37.

[15] Daniel Kahneman (2011): *Thinking, Fast and Slow*. Farrar, Straus and Giroux.

[16] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh & Peter Clark (2023): *Self-Refine: Iterative Refinement with Self-Feedback*. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt & S. Levine, editors: Advances in Neural Information Processing Systems, 36, Curran Associates, Inc., pp. 46534–46594. Available at https://proceedings.neurips.cc/paper_files/paper/2023/file/91edff07232fb1b55a505a9e9f6c0ff3-Paper-Conference.pdf.

[17] Theo Olausson, Alex Gu, Ben Lipkin, Cedegao Zhang, Armando Solar-Lezama, Joshua Tenenbaum & Roger Levy (2023): *LINC: A Neurosymbolic Approach for Logical Reasoning by Combining Language Models with First-Order Logic Provers*. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, p. 5153–5176, doi:10.18653/v1/2023.emnlp-main.313.

[18] Liangming Pan, Alon Albalak, Xinyi Wang & William Wang (2023): *Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning*. In Houda Bouamor, Juan Pino & Kalika Bali, editors: Findings of the Association for Computational Linguistics: EMNLP 2023, Association for Computational Linguistics, Singapore, pp. 3806–3824, doi:10.18653/v1/2023.findings-emnlp.248. Available at https://aclanthology.org/2023.findings-emnlp.248/.

[19] Aske Plaat, Annie Wong, Suzan Verberne, Joost Broekens, Niki van Stein & Thomas Back (2024): *Multi-Step Reasoning with Large Language Models, a Survey*. arXiv preprint arXiv:2407.11511, doi:10.48550/arXiv.2407.11511.

[20] Nils Reimers & Iryna Gurevych (2019): *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), Association for Computational Linguistics, doi:10.18653/v1/d19-1410.

[21] Abulhair Saparov & He He (2022): *Language models are greedy reasoners: A systematic formal analysis of chain-of-thought*. arXiv preprint arXiv:2210.01240, doi:10.48550/arXiv.2210.01240.

[22] Parshin Shojaee, Iman Mirzadeh, Keivan Alizadeh, Maxwell Horton, Samy Bengio & Mehrdad Farajtabar (2025): *The Illusion of Thinking: Understanding the Strengths and Limitations of Reasoning Models via the Lens of Problem Complexity*. arXiv preprint arXiv:2506.06941, doi:10.48550/arXiv.2506.06941.

[23] Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Garriga-Alonso et al. (2022): *Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models*. arXiv preprint arXiv:2206.04615, doi:10.48550/arXiv.2206.04615.

[24] Oyvind Tafjord, Bhavana Dalvi & Peter Clark (2021): *ProofWriter: Generating Implications, Proofs, and Abductive Statements over Natural Language*. In Chengqing Zong, Fei Xia, Wenjie Li & Roberto Navigli, editors: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, Association for Computational Linguistics, Online, pp. 3621–3634, doi:10.18653/v1/2021.findings-acl.317. Available at https://aclanthology.org/2021.findings-acl.317/.

[25] Team Gemini, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican et al. (2023): *Gemini: a family of highly capable multimodal models*. arXiv preprint arXiv:2312.11805, doi:10.48550/arXiv.2312.11805.

[26] Simon Vandevelde, Jeroen Jordens, Bart Van Doninck, Maarten Witters & Joost Vennekens (2022): *Knowledge-Based Support for Adhesive Selection*. In Georg Gottlob, Daniela Inclezan & Marco Maratea, editors: *Logic Programming and Nonmonotonic Reasoning*, Springer International Publishing, Cham, pp. 445–455, doi:10.1007/978-3-031-15707-3_34.

[27] Simon Vandevelde, Joost Vennekens, Jeroen Jordens, Bart Van Doninck & Maarten Witters (2024): *Knowledge-Based Support for Adhesive Selection: Will it Stick?* Theory and Practice of Logic Programming 24(3), p. 560–580, doi:10.1017/s1471068424000024.

[28] Lars Vermeulen, Simon Vandevelde & Joost Vennekens (2022): *FOLINT: static code analysis for FO(.)*. Available at https://gitlab.com/krr/IDP-Z3/-/tree/main/folint.

[29] Stijn Voet, Christian Fleiner & Simon Vandevelde (2025-04-18): *Towards Knowledge Formalization for Domain Experts using LLMs*. In: *Proceedings of HHAI 2025.*

[30] Zhaofeng Wu, Linlu Qiu, Alexis Ross, Ekin Akyürek, Boyuan Chen, Bailin Wang, Najoung Kim, Jacob Andreas & Yoon Kim (2024): *Reasoning or Reciting? Exploring the Capabilities and Limitations of Language Models Through Counterfactual Tasks*. Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), Association for Computational Linguistics, Mexico City, Mexico, pp. 1819–1862, doi:10.18653/v1/2024.Naacl-Long.102. Available at https://aclanthology.org/2024.naacl-long.102/,https://doi.org/10.18653/v1/2024.naacl-long.102.

[31] Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee & Wynne Hsu (2024): *Faithful Logical Reasoning via Symbolic Chain-of-Thought*. In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, p. 13326–13365, doi:10.18653/v1/2024.acl-long.720.

[32] Zhun Yang, Adam Ishay & Joohyung Lee (2023): *Coupling Large Language Models with Logic Programming for Robust and General Reasoning from Text*. In: *Findings of the Association for Computational Linguistics: ACL 2023*, Association for Computational Linguistics, doi:10.18653/v1/2023.findings-acl.321.

[33] Wanjun Zhong, Siyuan Wang, Duyu Tang, Zenan Xu, Daya Guo, Jiahai Wang, Jian Yin, Ming Zhou & Nan Duan (2021): *AR-LSAT: Investigating Analytical Reasoning of Text*. *CoRR* abs/2104.06598. arXiv:2104.06598.

An LLM + ASP Workflow for Joint Entity-Relation Extraction

Trang Tran
New Mexico State University
New Mexico, USA
ttran@nmsu.edu

Trung Hoang Le
New Mexico State University
New Mexico, USA
trungle@nmsu.edu

Huiping Cao
New Mexico State University
New Mexico, USA
hcao@nmsu.edu

Tran Cao Son
New Mexico State University
New Mexico, USA
stran@nmsu.edu

Joint entity-relation extraction (JERE) identifies both entities and their relationships simultaneously. Traditional machine-learning based approaches to performing this task require a large corpus of annotated data and lack the ability to easily incorporate domain specific information in the construction of the model. Therefore, creating a model for JERE is often labor intensive, time consuming, and elaboration intolerant. In this paper, we propose harnessing the capabilities of generative pretrained large language models (LLMs) and the knowledge representation and reasoning capabilities of Answer Set Programming (ASP) to perform JERE. We present a generic workflow for JERE using LLMs and ASP. The workflow is generic in the sense that it can be applied for JERE in any domain. It takes advantage of LLM's capability in natural language understanding in that it works directly with unannotated text. It exploits the elaboration tolerant feature of ASP in that no modification of its core program is required when additional domain specific knowledge, in the form of type specifications, is found and needs to be used. We demonstrate the usefulness of the proposed workflow through experiments with limited training data on three well-known benchmarks for JERE. The results of our experiments show that the LLM + ASP workflow is better than state-of-the-art JERE systems in several categories with only 10% of training data. It is able to achieve a 2.5 times (35% over 15%) improvement in the Relation Extraction task for the SciERC corpus, one of the most difficult benchmarks.

1 Introduction

Named Entity Recognition (NER) and Relationship Extraction (RE) are classification tasks in Natural Language Processing (NLP) focused on identifying and labeling entities and relationships from unstructured data into predefined categories as discussed by [LSHL22]. Both tasks are useful in information extraction, knowledge graph creation, and question answering ([YWZ+24, LSHL22]). When both tasks are performed simultaneously by the same model, it is known as joint entity-relation extraction (JERE) as defined by [ZHL+17]. It is well-known that JERE is much harder than NER or ER. Traditionally, supervised models are most effective when trained on large sets of domain specific annotated data for NER and RE tasks (see, e.g., [VMA24]).

In recent years, Large Language Models (LLMs) such as Generative Pretrained Transformer (GPT) have been used in information extraction tasks with techniques such as instruction tuning ([WZZ+23]), transforming sequence labeling tasks into generation tasks ([WSL+23]), and augmenting datasets for finetuning ([SSCS24]). To the best of our knowledge, LLMs have not been used specifically for the

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 63–75, doi:10.4204/EPTCS.439.6

© T. Tran, T. H. Le, H. Cao, T. C. Son
This work is licensed under the
Creative Commons Attribution License.

JERE task. Nevertheless, [BBMD24] has shown that fine-tuning a GPT can enhance results in NER tasks with the improvements depending on the amount of data used to fine-tune.

This paper introduces a novel approach that combines LLMs with logic programming under answer set semantics (ASP), introduced by [GV90], to jointly identify entities and their relationships from unstructured text. This design leverages the vast knowledge base embedded in GPT, which has been pretrained on billions of data points across various domains. While GPT's initial capabilities are impressive, they are still prone to producing hallucinations, which are falsified information presented as fact about real-world subjects as discussed by [TZJ+24]. In this paper, we propose using ASP and domain specific knowledge, whenever it is available, to mitigate false predictions generated by the LLM.

The main contributions of this work are as follows.

- A workflow that is elaboration tolerant by exploiting GPT's corpus and broad applicability along with ASP's flexibility for use in JERE tasks. It is a simple, but effective, workflow that shows how symbolic knowledge representation can further improve upon generative outputs from LLMs.
- A modular prompt template that can be used for JERE tasks across domains.
- An experimental evaluation demonstrating the superiority of the proposed approach compared to two state-of-the-art methods, using three commonly used benchmark datasets for JERE.

The next section presents the necessary background and related works in JERE and ASP. Section 3 details our approach. Section 4 describes our experiments and their results. We conclude the paper in Section 5.

2 Background and Related Works

An ASP program consists of rules of the form "*head* $\leftarrow$ *body*" where *head* is an atom and *body* is a conjunction of atoms or default negations of atoms in a first order language. Intuitively, a rule states that if the *body* is true then the *head* must be true. Answer set semantics of a logic program is define by [GV90] and can be computed efficiently using answer set solvers such as `clingo`[1]. In this paper, we employ ASP with extended syntax such as choice atoms, aggregate atoms, and constraints that has become the standard of logic programming language and implemented in most answer set solvers. It is worth noticing that, recently, ASP has been used to enhance the logical reasoning and accuracy of GPT outputs in code generation tasks and spatial reasoning as discussed in [KSB+24] and [WSK24]. The consistency checking step in this work is inspired by the system ASPER (see below).

The literature on NER, ER, and JERE spans a wide range of methods, from traditional rule-based approaches to more recent machine learning and deep learning techniques. The surveys by [LSHL22] and [YWZ+24] mainly focus on NER. Early work in NER, ER, and JERE often relied on handcrafted rules and annotators to identify entities and relationships, which limited scalability and accuracy. With the rise of supervised learning, models like Conditional Random Fields (CRFs) and Support Vector Machines (SVMs) were introduced, allowing for more flexible and data-driven extraction. Deep learning approaches, especially those based on transformer architectures (e.g., BERT, RoBERTa), have significantly advanced JERE by leveraging pretrained contextual embeddings. These models have shown superior performance in a variety of domains, including biomedical text mining, legal document analysis, and social media content. Most recently, prompt engineering frameworks have been implemented to take advantage of pretrained large language models. InstructUIE[WZZ+23] uses multi-task instruction and fine-tuning to identify named entities (without classifying their types) and to extract relationships between entities separately, rather than jointly detecting entities, their types, and their relationships. RIIT-IE[GSJ+24] attempts to distill noise from true positives when detecting entities and their relationships, using iterative

[1] `https://potassco.org`

and hierarchical prompt engineering. Among prompt-only methods, its framework achieves the best performance we have seen. However, RIIT-IE employs significantly more complex prompting techniques. It uses a modular system where data pass through multiple layers, with different prompts applied at each layer to progressively narrow down the correct answers.

Despite the advances, challenges still remain, such as handling ambiguous entities, identifying novel relationships, and extracting information from noisy or unstructured data. As a result, ongoing research is focused on enhancing model generalization, developing domain-specific models, and incorporating external knowledge sources to further improve the accuracy and robustness of joint entity relationship extraction.

To the best of our knowledge, the system ASPER, developed by [LCS23], performs better than state-of-the-art JERE systems when it was introduced. It is shown in that paper that domain specific knowledge can be exploited effectively in reducing the amount of training data and in increasing the model performance. ASPER employs ASP to improve the learning process of neural network models. ITER, the most recent introduction to the JERE landscape by [HBG24], is currently the best system for JERE. It is an encoder-only, transformer-based model. ITER's performance, however, depends on the amount of data used in its training.

3 A Lightweight LLM + ASP Approach

3.1 Overview of the Proposed Method

We propose a lightweight workflow to conduct effective JERE. The framework consists of two main components (Figure 1): (*i*) a generic prompt template for JERE, given the domain and annotation guideline; and (*ii*) a consistency checker that is written in ASP. The template aims at asking a LLM, GPT or Gemini in our study, to extract entities and relations from the domain. Use of a retrained LLM can take advantage of the knowledge that is learned in the model and save time on training another new machine learning model. At the same time, it is well known that LLMs such as GPT produce hallucinations (see, e.g., ([PDB24])). This means that the entities and relationships returned from an LLM model may have both false positives and false negatives. To help improve the quality of LLM output, we design a novel strategy to verify the consistency of the output and eliminate inconsistent outputs. More concretely, in step (*ii*), the output of an LLM model is then provided to an ASP solver together with available domain-specific knowledge, called *type specification*, for consistency checking.

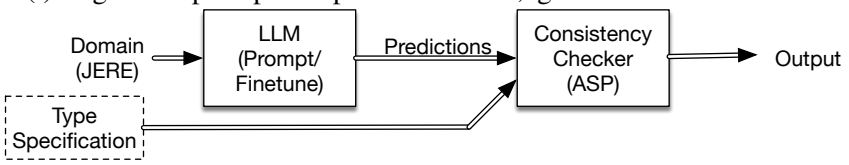


Figure 1: LLM + ASP Workflow for JERE

We next detail the design of the prompt template (Section 3.3) and the ASP program for consistency checking (Section 3.4).

3.2 Pre-study

To create an effective and generic template for JERE, we conducted a comprehensive study on the state-of-the-art prompting techniques for entity-relation extraction tasks. Techniques such as In-Context-Learning, Chain-Of-Thought, zero-shot, and few-shot prompting ([SIB$^+$24]) were tested to see which (or a combination of them) would yield good results with respect to our task. We employed GPT-3.5 and

used the ConLL04 as a sample domain to conduct this preliminary study. The experiments showed that the following four techniques in combination resulted in a 6% increase in the F1-macro score of both entity and relationship extraction tasks without fine-tuning. They are:

- Giving GPT specific context by clearly defining a domain and the role it will take on.
- The use of one-shot prompting by including one example.
- Addition of constraints and definitions for what is considered an entity or relationship in the confines of our dataset.
- Answer engineering in which we defined the output key specifications in JSON format.

We used the prompt building techniques we learned through this pre-study to inform our prompt engineering in the next section.

3.3 Prompt Engineering

For the JERE task, a prompt template needs to define the domain, experience, context, output keys, and one example. Since our goal is to create a JERE system that can work with arbitrary domains, we create a generic base template for the JERE task that can be augmented with domain-specific information. In this sense, our template is similar to the modular template used in PromptNER, introduced by [AL23].

A *Domain* is a general term to narrow the field in which the LLM agent is asked to focus. *Experience* refers to how much experience the GPT agent has within the given domain. The *Context* includes general definitions for what is considered an entity or relationship, the types and how to annotate the text for the specified entity and relationship types. It is derived directly from the annotation guidelines for each dataset. The domain, experience, and context are assigned in the system prompt. This gives the GPT agent background knowledge of the task and domain.

Output Keys refer to the specific keys used for evaluation. The output keys and one example are stated in the user prompt and give more specificity to the LLM agent. All of the user-defined categories above are informed by the annotation guides supplied by each dataset.

Example 3.1 *Below is an example of how a dataset has been broken down into the different categories and the full prompt:*

- ***Domain:*** *"journalism and news"*
- ***Experience:*** *"You have an M.Sc. degree in linguistics and substantial background working to annotate entities and relationships using your knowledge of syntax and semantics."*
- ***Context:*** *"Only classify entity types as either location, organization, people, or other. Output 'Loc' for location, 'Peop' for people, 'Org' for organization and 'Other' for other. Only classify relationship types as either organization based in, located in, live in, work for, or kill. Output 'OrgBased_In' type for organization based in, 'Located_In' for located in, 'Live_In' for live in, 'Work_For' for work for, and 'Kill' for kill."*
- ***Output Key:*** *"entities": ["entity":, "type":], "relationships": ["subject":, "object":, "type":]*
- ***Example:*** *"Input: "Andrew Jackson, born March 15, 1767, in Waxhaw settlement.", Output:{"Entities":[{"Entity": "Andrew Jackson", "Type":"Peop"},{"Entity": "March", "Type": "Other"},{"Entity": "Waxhaw", "Type": "Loc"}], "Relationships":[{"Subject": "Andrew Jackson","Object": "Waxhaw", "Type": "Live_In"}}"*

3.3.1 Base Prompt Template

Our prompt template consists of two components, system and user. They are defined as follows.
System:

"You are a natural language processing researcher working in the {DOMAIN} domain. {EXPERIENCE} Your job is to extract entities from the excerpts of texts given. In this domain, an entity is an object, set of objects or abstract notion in the world that has its own independent existence. Entities specify pieces of information or objects within a text that carry particular significance. In your work, you will only extract specific types of entities and relationships. The types of entities and relationships are defined here. {CONTEXT}"
User:

"Give me the entities from the following text. Do not include any explanations, only provide RFC8259 compliant JSON response without deviation. Do not include '\n' (`newline`) in the output. The keys for the output JSON should be {OUTPUT_KEYS}. Do not use any other keys for the JSON response. Ensure that you are outputting the entire entity and its type. Here is one example: {EXAMPLE} Evaluate this text: {TEXT}"

As we have seen with GPT's, the more specific a prompt is, the better the results, but it is time consuming to consider how to engineer a prompt for every situation. This template for both system and user prompts allowed us to generalize the task even when the datasets are not related. We must still determine what appropriate information should be included in a prompt, but the base template gives us guidance on what type of information is relevant and needed.

3.4 Consistency Checking Using Answer Set Programming

To eliminate potential false predictions from the output of the LLM, we propose a verifying step, termed as *consistency checking*. This step takes advantage of the facts that the LLM's output is essentially a collection of atoms, and thus, can be easily manipulated via rules. The idea of utilizing ASP to conduct consistency check originated from ASPER [LCS23]. However, the available data structure in this work is completely different from that used in ASPER and therefore, the code in this work is different from that used in ASPER and, we believe, is much easier to understand.

3.4.1 ASP Program for Consistency Checking

We describe the program, denoted by Π_C, that is the main ingredient of the consistency checking step. This program takes the output of the GPT encoded as a collection of facts of the forms

- `atom(ent(S,E,T))`: E is an entity of the type T in the sentence S; and
- `atom(rel(S,E,F,R))`: relation of type R between entities E and F in the sentence S.

Optional inputs of the program include

- Type specification of the form `type_def(R,T,V)`: relation of type R is between entities of the types T and V;
- Ground truth of the form `ent(S,E,T)` and `rel(S,E,F,R)` whose meaning is similar to that of `atom(ent(_))` and `atom(rel(_))`, respectively.

The program defines the following predicates:

- `false_declaration(S,E,F,R)`: at least one of the entities, E or F, of the relation R does not appear in the entity list;
- `ok_type(S,E,F,R)`: the type of the relation R between E and F matches its specification; and

- `has_declaration(R)`: the type of the relation R is specified.

The predicate `false_declaration` encodes relations that are *inconsistent* with the set of entities while `ok_type` reports relations that are *consistent* with the type specification. These predicates are defined by the following rules:

Listing 1: ASP Program for Consistency Checking

```
1  false_declaration(S,E,F,R):-  atom(rel(S,E,F,R)),
2      1{not atom(ent(S,E,_)); not atom(ent(S,F,_))}.
3  has_declaration(R) :- type_def(R, _, _).
4  ok_type(S,E,F,R):-atom(rel(S,E,F,R)),atom(ent(S,E,T)),atom(ent(S,F,V)),
5      1{type_def(R,T,V); not has_declaration(R)}.
```

We denote the above set of rules by Π^1_C. The first rule (Lines 1-2) defines when a relation has false declaration. The atom `1{not atom(ent(S,E,_)); not atom(ent(S,E',_))}` (Line 2) indicates that at least one of the atoms `atom(ent(S,E,_))` or `atom(ent(S,F,_))` is not contained in the output of the model, i.e., either `E` or `F` was not detected as an entity by the LLM. The rule defining `ok_type(S,E,F,R)` (Lines 4–5) says that the type of the relation (`R`) is appropriate given the type specification or the type of the relation `R` is unspecified. This allows for the program to be used with or without type specification. Line 3 is used to indicate that domain-specific information is available.

Given the model output O and set of type specification atoms D, it is easy to see that the program $\Pi^1_C \cup O \cup D$ has a unique answer set $O \cup D \cup W$ where W is a collection of atoms of the form `false_declaration(s,e,f,r)`, `has_declaration(r)`, and `ok_type(s,e,f,r)`. Note that if $D = \emptyset$, i.e., type specification is not available, then all RE predictions have the correct type, and thus, are acceptable. We consider `rel(s,e,f,r)` as invalid if W contains `false_declaration(s,e,f,r)` or does not contain `ok_type(s,e,f,r)` and remove it from the output of the model.

The next set of rules can be used for computing the various components needed for computing the F1-scores (macro-F1 and micro-F1). When the ground truth is not provided, these rules are not activated and will not change the content of the answer set of $\Pi^1_C \cup O \cup D$. In the code, `#count` refers to the aggregate counting the number of elements in a set specified between the brackets { and }. The rules defining the predicates `r_true_pos/4` (Lines 7–8) and `r_false_pos/4` (Lines 9–11) remove predictions with incorrect type or false declaration from consideration. The meaning of the other predicates is easily understood and is therefore omitted for brevity.

Listing 2: Computing True/False Positive/Negative and F1-score

```
6  in_set(S):-atom(ent(S, _, _)). in_set(S):-atom(rel(S, _, _, _)).
7  r_true_pos(S,E,F,R):- atom(rel(S,E,F,R)),
8      ok_type(S,E,F,R),rel(S,E,F,R).
9  r_false_pos(S,E,F,R):- atom(rel(S,E,F,R)), ok_type(S,E,F,R),
10     not false_declaration(S,E,F,R), not rel(S,E,F,R).
11 r_false_neg(S,E,F,R):- rel(S,E,F,R), in_set(S), not atom(rel(S,E,F,R)).
12 r_true_p_cnt(C,T):-type_of_r(T),C=#count{S,E,F:r_true_pos(S,E,F,T)}.
13 r_false_p_cnt(C,T):-type_of_r(T),C=#count{S,E,F:r_false_pos(S,E,F,T)}.
14 r_false_n_cnt(C,T):-type_of_r(T),C=#count{S,E,F:r_false_neg(S,E,F,T)}.
15 e_true_pos(S,E,T):-ent(S,E,T), atom(ent(S,E,T)).
16 e_false_pos(S,E,T):- atom(ent(S,E,T)),not ent(S,E,T).
17 e_false_neg(S,E,T):-ent(S,E,T),in_set(S), not atom(ent(S,E,T)).
18 true_p_cnt(C,T):-type_of_ent(T),C=#count{S,E:e_true_pos(S,E,T)}.
19 false_p_cnt(C,T):-type_of_ent(T),C=#count{S,E:e_false_pos(S,E,T)}.
20 false_n_cnt(C,T):-type_of_ent(T),C=#count{S,E:e_false_neg(S,E,T)}.
```

4 Experimental Evaluation

4.1 Experimental Settings

Python code was implemented using Python 3.10 and OpenAI SDK version 1.57.0 and performed on a MacBook Pro with an Apple M3 Max chip. The fine-tuning and JERE tasks were run on OpenAI's servers and call the gpt-4o-2024-08-06 model [Ope24], referred to as **GPT** from now on. Specifically, we use the Batch and Fine-Tuning APIs from OpenAI. For the ensemble experiment, we use Google's Gemini Flash 1.5 [Goo24], referred to as **Gemini**, and the google-generativeai API version 0.8.3. The ASP solver is `clingo` 5.4.0 [GKKS14]. Source code and execution instruction related to the project can be found at the github [Tra25].

Data. Our work focuses on joint entity and relation extraction (JERE) identifying entities with their types and predicting relations between them within a single sentence. Therefore, we select the following benchmarks for our experiment:

- **CoNLL04** ([EU20, RY04, GSA16, WL20]): This dataset contains a total of 1,437 sentences retrieved from newspaper clippings and resides in the 'news and journalism' domain. It differentiates between 4 types of entities (`people, organization, location`, and `other`) and 5 types of relationships (`live_in, located_in, kill, orgbased_in`, and `work_for`).
- **SciERC** ([EU20, LHOH18]): This dataset contains 2,412 sentences from scientific abstracts and differentiates between 6 types of entities (`task, method, metric, material, otherScientifcTerm`, and `generic`) and 7 relationships (`compare, part-of, conjunction, evaluate-for, feature-of`, and `used-for, hyponym-of`).
- **ADE** [GMR$^+$12]: it contains 4,272 annotated documents from the 'health and drug' domain and differentiates between 2 types of entities (`drug` and `adverse-effect`) and one relationship (`adverse-effect`).

We note that there are other well-known benchmarks such as the TACRED, REFinD, SemEval-2010 Task 8 and DocRED datasets[2] that were used by some entity/entity-relation extraction systems. However, TACRED, SemEval-2010 and REFinD are designed to annotate entity pairs and their relationships within individual sentences, and hence, they may overlook other entities in the sentence, limiting their suitability for full entity extraction. DocRED consists of multi-sentence instances where the same entity can appear in different forms and locations within an instance, requiring entity resolution before applying the JERE task.

We note that there are domains rich in type specification such as the CoNLL04 domain. For example, the following relationships between types of the relations and entities were introduced by [LCS23]:

Listing 3: Type Specification CoNLL04

```
21 type_def("located_in","loc","loc"). type_def("live_in","peop","loc").
22 type_def("orgbased_in","org","loc").type_def("work_for","peop","org").
23 type_def("kill", "peop", "peop").
```

For the SciERC dataset, we derive a set of type specifications for this domain given the set of entities. Given the intuitive meaning of the entity types in the domain, we consider the following possible combinations of the `part-of` relation:

Listing 4: Type Specification SciErc

```
24 type_def("part-of","task","task").
```

[2] `https://nlp.stanford.edu/projects/tacred/`, `https://refind-re.github.io`, `https://arxiv.org/pdf/1911.10422`, `https://arxiv.org/pdf/1906.06127`

```
type_def("part-of","generic","generic").
type_def("part-of","material","material").
type_def("part-of","otherscientificterm","otherscientificterm").
type_def("part-of","metric","metric").
type_def("part-of","method","method").
type_def("part-of","otherscientificterm","method").
type_def("part-of","generic","method").
type_def("part-of","method","generic").
type_def("part-of","task","method").
```

The complete type specification for this domain can be found in [Tra25]. The ADE dataset has only two types of entities and thus no type specification is added.

Data processing. We preprocessed each raw dataset to extract full sentences and paragraphs for LLM input, rather than tokenized word lists. Our LLM request also specifies a human-readable output, rather than a list of indices or entity spans.

Baselines for comparison. Two competitors, (i) **ASPER** by [LCS23] and (ii) **ITER** by [HBG24], were chosen as baselines to compare with our proposed method. ASPER utilizes ASP to improve its quality of prediction and ITER has shown to outperform most other joint ER extraction techniques. We also implemented a variation of our workflow by replacing the ChatGPT LLM with an ensemble of LLMs, as ensembles generally yield better results than individual models. The ensemble consists of two LLM agents: the fine-tuned ChatGPT and a Gemini agent. Both agents are tasked with auditing the results, and if they both agree on an entity e of type t, that entity is included in the output. In reporting the results of this study, we refer to the ensemble of LLMs as **Ensemble**, and the ensemble with the ASP consistency checker as **Ensemble + ASP**. In all the result tables, we use **E** to represent entity and **ER** to represent entity-relationship.

Evaluation metrics. We use F_1-micro and F_1-macro scores to evaluate the model's performance on entities (NER) and entity-relation (ER) tasks. F_1-micro is calculated using the total true positives, false negatives and false positives. F_1-macro is the unweighted average of each class type's F_1 score. The formula for F_1 is $\frac{2TP}{2TP+FP+FN}$. It is generally accepted that systems with better F_1-macro score are considered "better."

4.1.1 Default Setting of The LLM+ASP Workflow

By default, we used a fine-tuned GPT agent, the gpt-4o-2024-08-06 model [Ope24], for the JERE outputs with the ASP consistency checker.

The prompt utilizes one-shot prompt. Each dataset's prompt was specific to that dataset by using the annotation guidelines given in the corpus' accompanying papers. For the fine-tuning step, we simulate a low-resource setting. Each dataset is originally split into training (65%), validation(15%) and test sets(20%). We randomly selected 10% of the original training data and 10% of the original validation data to fine-tune the model, using them for training and validation respectively. Each fine-tuned model was specific to its dataset. The hyper-parameters were consistent across all datasets, with 5 epochs, a batch size of 1, and a learning rate multiplier of 2.

The ASP consistency checker uses the ASP program, Π_C, detailed in the previous section that is also independent from the domain (code see [Tra25]). Domain specific information in the form of type specification is provided as an optional input to this program.

Dataset	One-shot prompt				Fine-tuned GPT			
	F1-Micro		F1-Macro		F1-Micro		F1-Macro	
	E	ER	E	ER	E	ER	E	ER
CoNLL04	73.29	44.78	67.42	48.42	80.27	58.82	74.59	57.31
SciErc	42.83	7.89	35.56	7.37	61.70	26.55	60.94	22.94
ADE	88.30	37.28	88.75	37.28	90.32	82.84	90.84	82.84

Table 1: One-shot prompt vs. Fine-tuned (E: entity; ER: entity-relationship; No ASP Checking)

Given that the fine-tuned models with the randomly chosen 10% of training data consistently outperforms the one-shot prompt (Table 1), we used the fine-tuned models throughout the rest of this paper. Additionally, because the models do not provide deterministic responses and may produce hallucinations, we run each model three times to obtain a more robust assessment and report the averaged results.

4.1.2 Training Time and Model Sizes

Most of the computational load of our proposed method is handled by OpenAI's servers. Fine-tuning GPT-4o on 400 training samples for the ADE dataset for 5 epochs takes ≈15 minutes, with evaluation of the full test set taking an additional 15 minutes. Similar running time is observed on the other datasets. The computation of the ASP consistency checker is efficient and nearly negligible, requiring only ∼10 milliseconds to process all predictions per dataset.

Regarding scalability, the main computational cost lies in fine-tuning and prediction. Fine-tuning scales linearly with data size and the number of epochs, while prediction scales linearly with the number of words, as it operates at the sentence level. The ASP consistency checker adds negligible overhead.

We would also report the sizes of the model utilized by our approach and the baselines. Our approach is based on a GPT agent gpt-4o-2024-08-06 model, which has approximately 1.76 trillion parameters. For comparison, the ASPER model uses around 110 million fixed (pretrained) parameters and approximately 20,000 trainable parameters across all datasets. The ITER model has a total of 393 million parameters for all datasets. The model sizes show one limitation of our approach in that it utilizes larger models compared with the baseline.

4.2 LLM + ASP vs. State-of-the-Art Systems

This section shows the effectiveness of our proposed LLM + ASP workflow using the default setting stated in Section 4.1.1 when compared with other baselines. The training data that is used in LLM fine-tuning is randomly chosen 10% of the original training set (default setting). For fair comparison, for both ASPER and ITER, we also used 10% of the original training data. For ITER, the 10% training data is the same as that used to fine-tune the LLM model. For ASPER, we use the authors' chosen 10% data to be consistent with their configuration.

Table 2 shows the overall results. Boldface numbers indicated systems with the best score in the corresponding category. As can be seen, our workflow is comparable to the state-of-the-art supervised models in ER. It consistently outperforms ASPER by [LCS23] in the ER task. On the SciErc dataset, it excels over ASPER by 20% raw score where GPT+ASP can achieve 35.37% F_1-macro while ASPER is at 16.06% F_1-macro.

Notably, existing state-of-the-art methods perform poorly on the SciERC dataset. Surprisingly, our workflow outperforms ITER by more than 25% raw score. We attribute this improvement to the ASP

Method	CoNLL04				SciErc				ADE			
	F1-Micro		F1-Macro		F1-Micro		F1-Macro		F1-Micro		F1-Macro	
	E	ER	E	ER	E	ER	E	ER	E	ER	E	ER
GPT+ASP	80.45	60.51	74.79	**58.91**	**62.32**	**38.23**	61.55	**35.37**	**90.40**	**83.89**	**90.91**	**83.89**
Ens.+ASP	80.29	**60.54**	74.44	58.59	**62.32**	37.23	**61.64**	**35.37**	89.53	82.21	90.08	82.21
ASPER	**81.25**	52.41	**75.90**	53.27	60.34	21.73	59.10	16.06	86.60	75.30	86.93	75.30
ITER	70.81	34.37	63.15	27.58	56.07	10.53	55.46	10.00	86.49	75.70	87.10	75.70

Table 2: Performance comparison of different systems (E: entity, ER: entity-relationship)

consistency checker, which reduces FP and, as a result, enhances the quality of entity-relation resolutions. We want to note that when trained on 100% of the training data, ITER outperforms our GPT+ASP, that used only the randomly chosen 10% of the original training data, by 7% raw score.

4.3 Ablation Studies

This set of experiments is to examine the effect of two components (1) the ASP consistency checker and (2) the ensemble of the LLMs.

Methods	CoNLL04				SciErc				ADE			
	F1-Micro		F1-Macro		F1-Micro		F1-Macro		F1-Micro		F1-Macro	
	E	ER	E	ER	E	ER	E	ER	E	ER	E	ER
GPT+ASP	80.45	60.51	74.79	58.91	62.32	38.23	61.55	**35.37**	90.40	83.89	90.91	83.89
GPT	80.27	58.82	74.59	57.31	61.70	26.55	60.94	**22.94**	90.32	82.84	90.84	82.84
Ens.+ASP	80.29	60.54	74.44	58.59	62.32	37.23	61.64	**35.37**	89.53	82.21	90.08	82.21
Ensemble	80.51	58.15	75.07	56.75	61.20	26.14	60.59	**25.15**	90.10	82.06	60.41	82.06

Table 3: Results for ablation study (E: entity, ER: entity-relationship); Ens.: GPT and Gemini Ensemble

Effect of ASP consistency checker. The first experiment demonstrates the contribution of the ASP consistency checker to our workflow (see Table 3). Boldface numbers highlight the scores in SciERC dataset, the most difficult dataset for JERE. We compare the outputs (entities and relationships) from our default workflow, GPT+ASP (Row 1, Table 3), with those from the fine-tuned GPT alone (Row 2, Table 3), as well as the results from Ens.+ASP (Row 3) and the ensemble model alone (Row 4). As can be observed, both the F1-macro and F1-micro scores with the ASP consistency checker (Rows 1 and 3) improve upon the corresponding version without the ASP consistency checker (Rows 2 and 4) for ER, sometimes more than 30% (SciErc dataset). This demonstrates the effectiveness of the ASP checker in the process when domain specifications are available. In ADE, we do not see a large increase in the ER scores since there is only one relationship type to extract and thus less to reduce based on the domain knowledge.

We conducted a more detailed analysis of this improvement by examining how many entities and entity relationships are truly or falsely reported as positive. Table 4 presents the numbers for GPT+ASP and GPT alone. The results show that, across all three datasets, the number of falsely reported positive entity-relationships are reduced with the use of the ASP consistency checker. Datasets with more type-specifications, like SciErc, benefited most from the consistency checker - going from 713 FP values to 482.

Dataset	GPT						GPT+ASP					
	E			ER			E			ER		
	TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN
CoNLL04	881	258	176	262	224	144	883	256	174	264	203	142
SciErc	1003	575	670	244	713	639	1004	574	640	339	482	614
ADE	991	98	114	571	112	125	992	98	113	579	105	117

Table 4: Effect of ASP to improve FP (Randomly chosen 10% Training Data). (TP: True Positive, FP: False Positive, FN: False Negative; E: entity, ER: entity-relationship)

Effect of LLM ensemble. The second experiment examines whether the LLM ensemble helps improve an individual LLM agent. The result is reported in Table 3. As it turns out, the ensemble, in its current use, does not perform better than the single GPT agent, with or without the ASP checker. This can be seen in the results in Row 2 vs. Row 4 (GPT vs. Ensemble) and Row 1 vs. Row 3 (GPT + ASP vs. Ensemble + ASP). The reason for this reduced performance is that TP entities, detected by GPT, are removed from consideration which, ultimately, reduces the F_1-macro/micro scores.

4.4 Effect of Amount of Training Data

Table 5 shows the results of our default workflow with different versions of GPT, fine-tuned on 5%, 10%, and 15% of training data, respectively. These small percentages of training data are all randomly chosen. For each setting, the LLM model is fine-tuned three times and the reported number is the average of the results from the three fine tuned models.

	5% TD+ ASP checker				10% TD + ASP checker				15% TD + ASP checker			
	F1-Micro		F1-Macro		F1-Micro		F1-Macro		F1-Micro		F1-Macro	
	E	ER	E	ER	E	ER	E	ER	E	ER	E	ER
C	77.68	58.07	72.52	58.64	80.45	60.51	74.79	58.91	80.20	57.71	73.58	55.31
S	59.14	32.46	59.15	31.14	62.32	38.23	61.55	35.37	64.16	40.63	64.10	35.05
A	90.13	79.61	90.61	79.61	90.40	83.89	90.91	83.89	90.73	84.00	91.16	84.00

Table 5: Results different percentages of training data on fine-tuned ChatGPT model. (E: entity; ER: entity-relationship; TD: Training Data; C: CoNLL04; S: SciErc; A: ADE)

Overall, the workflow performs better with more data with some exception. Its performance seems to be domain-dependent. We can observe distinct improvement from 5% to 10% from for each dataset. However, in the ADE results we can see improvements only in the ER task - with there being minimal difference between the models fine-tuned on 10% and 15%. The overall scores are better for SciErc with the exception of the F_1-macro score for the ER task between the 15% and 10% models. In the CoNLL04 dataset, we see a reduced score when comparing 10% and 15% data.

5 Conclusion

In this paper, we propose a generic workflow for joint entity-relation extraction using LLMs and ASP. The workflow is used to perform the JERE task on arbitrary domains. Due to the LLM's capability in natural language understanding, our system can perform the JERE task on *unannotated text*, which sets it

apart from contemporary systems that require large amounts of annotated tokenized text. The workflow can exploit domain-specific information, when available, to improve its performance. In addition, our approach offers greater flexibility and scalability, as it can adapt to new domains with minimal additional fine-tuning. We demonstrate the usefulness of the proposed workflow through experiments with limited training data on three well-known benchmarks for JERE. The results of our experiments show that the LLM+ASP workflow is better than state-of-the-art JERE systems in several categories. In the near future, we plan to explore using this workflow to extract knowledge graphs as they consist of entities and relations.

Acknowledgment

This work has been supported by NSF award #1914635. The first and last authors were also supported by NRC Grant 31310022M0038.

References

[AL23] D. Ashok & Z. C. Lipton (2023): *PromptNER: Prompting For Named Entity Recognition*. doi:10.48550/arXiv.2305.15444. arXiv:2305.15444.

[BBMD24] A. Bonfigli, L. Bacco, M. Merone & F. Dell'Orletta (2024): *From pre-training to fine-tuning: An in-depth analysis of Large Language Models in the biomedical domain*. Artificial Intelligence in Medicine, p. 103003, doi:10.1016/j.artmed.2024.103003. arXiv:2024.103003.

[EU20] M. Eberts & A. Ulges (2020): *Span-Based Joint Entity and Relation Extraction with Transformer Pre-Training*, pp. 2006–2013. doi:10.3233/FAIA200321.

[GKKS14] M. Gebser, R. Kaminski, B. Kaufmann & T. Schaub (2014): *Clingo = ASP + Control: Preliminary Report*. *14(4-5)*, doi:10.48550/arXiv.1405.3694. arXiv:1405.3694.

[GMR+12] H. Gurulingappa, A. Mateen, A. Roberts, J. Fluck, M. Hofmann-Apitius & L. Toldo (2012): *Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports*. Journal of Biomedical Informatics, pp. 885–892, doi:10.1016/j.jbi.2012.04.008.

[Goo24] GoogleAI (2024): *Gemini [Large Language Model]*. Available at `https://ai.google.dev/gemini-api/docs/models/gemini#gemini-1.5-flash`.

[GSA16] P. Gupta, H. Schütze & B. Andrassy (2016): *Table Filling Multi-Task Recurrent Neural Network for Joint Entity and Relation Extraction*, pp. 2537–2547.

[GSJ+24] H. Geng, C. Shi, X. Jiang, Z. Kong & S. Liu (2024): *An Entity Relation Extraction Framework Based on Large Language Model and Multi-Tasks Iterative Prompt Engineering*. IEEE International Conference on Systems, Man, and Cybernetics (SMC), doi:10.1109/SMC54092.2024.10831494.

[GV90] M. Gelfond & V.Lifschitz (1990): *Logic programs with classical negation*. MIT Press, Cambridge, USA.

[HBG24] M. Hennen, F. Babl & M. Geierhos (2024): *ITER: Iterative Transformer-based Entity Recognition and Relation Extraction*. Findings of the Association for Computational Linguistics: EMNLP 2024, pp. 11209–11223, doi:10.18653/v1/2024.findings-emnlp.655.

[KSB+24] A. Kalyanpur, K. K. Saravanakumar, V. Barres, J. Chu-Carroll, D. Melville & D. Ferrucci (2024): *LLM-ARC: Enhancing LLMs with an Automated Reasoning Critic*. doi:10.48550/arXiv.2406.17663. arXiv:2406.17663.

[LCS23] T. H. Le, H. Cao & T. C. Son (2023): *ASPER: Answer Set Programming Enhanced Neural Network Models for Joint Entity-Relation Extraction*. *TPLP*, pp. 765–781, doi:10.1017/S1471068423000297.

[LHOH18] Y. Luan, L. He, M. Ostendorf & H. Hajishirzi (2018): *Multi-Task Identification of Entities, Relations, and Coreference for Scientific Knowledge Graph Construction*. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pp. 3219–3232, doi:10.18653/v1/D18-1360.

[LSHL22] J. Li, A. Sun, J. Han & C. Li (2022): *A Survey on Deep Learning for Named Entity Recognition*. IEEE Transactions on Knowledge and Data Engineering, pp. 50–70, doi:10.1109/TKDE.2020.2981314. arXiv:2020.2981314.

[Ope24] OpenAI (2024): *GPT-4o [Large Language Model*. Available at `https://platform.openai.com/docs/models#gpt-4o`.

[PDB24] G. Perković, A. Drobnjak & I. Botički (2024): *Hallucinations in LLMs: Understanding and Addressing Challenges*, pp. 2084–2088. doi:10.1109/MIPRO60963.2024.10569238.

[RY04] D. Roth & W. Yih (2004): *A Linear Programming Formulation for Global Inference in Natural Language Tasks*. Proceedings of the Eighth Conference on Computational Natural Language Learning (CoNLL-2004) at HLT-NAACL 2004, pp. 1–8.

[SIB+24] S. Schulhoff, M. Ilie, N. Balepur, K. Kahadze, A. Liu, C. Si, Y. Li, A. Gupta, H. Han, S. Schulhoff, P. S. Dulepet, S. Vidyadhara, D. Ki, S. Agrawal, C. Pham, G. Kroiz, F. Li, H. Tao, A. S., H. Da Costa, S. Gupta, M. L. Rogers, I. Goncearenco, G. Sarli, I. Galynker, D. Peskoff, M. Carpuat, J. White, S. Anadkat, A. Hoyle & P. Resnik (2024): *The Prompt Report: A Systematic Survey of Prompting Techniques*. doi:10.48550/arXiv.2406.06608. arXiv:2406.06608.

[SSCS24] J. Santoso, P. Sutanto, B. Kelvianto Cahyadi & E. Irawati Setiawan (2024): *Pushing the Limits of Low-Resource NER Using LLM Artificial Data Generation*. In L. Ku, A. Martins & V. Srikumar, editors: Findings of the Association for Computational Linguistics: ACL 2024, Association for Computational Linguistics, Bangkok, Thailand, p. 9652–9667, doi:10.18653/v1/2024.findings-acl.575.

[Tra25] T. Tran (2025): *LLM+ASP Workflow Github Repository*. Available at `https://github.com/trangtran321/LLM_ASP_Workflow`.

[TZJ+24] S. M. T. Islam Tonmoy, S. M. M. Zaman, V. Jain, A. Rani, V. Rawte, A. Chadha & A. Das (2024): *A Comprehensive Survey of Hallucination Mitigation Techniques in Large Language Models*. doi:10.48550/arXiv.2401.01313. arXiv:2401.01313.

[VMA24] F. Villenaa, L. Mirandab & C. Aracenab (2024): *llmNER: (Zero\Few)-Shot Named Entity Recognition, Exploiting the Power of Large Language Models*. CSUR, doi:10.48550/arxiv.2406.04528. arXiv:2406.04528.

[WL20] J. Wang & W. Lu (2020): *Two Are Better than One: Joint Entity and Relation Extraction with Table-Sequence Encoders*. EMNLP, pp. 1706–1721, doi:10.48550. arXiv:2042.12345.

[WSK24] R. Wang, K. Sun & J. Kuhn (2024): *Dspy-based Neural-Symbolic Pipeline to Enhance Spatial Reasoning in LLMs*. doi:10.48550/arXiv.2411.18564. arXiv:2411.18564.

[WSL+23] S. Wang, X. Sun, X. Li, R. Ouyang, F. Wu, T. Zhang, J. Li & G. Wang (2023): *GPT-NER: Named Entity Recognition via Large Language Models*. doi:10.48550/arXiv.2304.10428. arXiv:2304.10428.

[WZZ+23] X. Wang, W. Zhou, C. Zu, H. Xia, T. Chen, Y. Zhang, R. Zheng, J. Ye, Q. Zhang, T. Gui, J. Kang, J. Yang, S. Li & C. Du (2023): *InstructUIE: Multi-task Instruction Tuning for Unified Information Extraction*. doi:10.48550/arXiv.2304.08085. arXiv:2304.08085.

[YWZ+24] M. Yan, L. Wang, R. Zhang, H. Cheng, W. Lam, Y. Shen & R. Xu (2024): *A Comprehensive Survey on Relation Extraction: Recent Advances and New Frontiers*. ACM Computing Surveys, Volume 56, Issue 11, pp. 1–39, doi:10.1145/3674501.

[ZHL+17] S. Zheng, Y. Hao, D. Lu, H. Bao, J. Xu, H. Hao & B. Xu (2017): *Joint entity and relation extraction based on a hybrid neural network*. Neurocomputing, pp. 59–66, doi:10.1016/j.neucom.2016.12.075.

VECSR: Virtually Embodied Common Sense Reasoning System

Alexis R. Tudor
University of Texas at Dallas, USA
alexisrenee1@gmail.com

Joaquín Arias
CETINIA
Universidad Rey Juan Carlos, Spain
joaquin.arias@urjc.es

Gopal Gupta
University of Texas at Dallas, USA
gupta@utdallas.edu

The development of autonomous agents has seen a revival of enthusiasm due to the emergence of LLMs, such as GPT-4o. Deploying these agents in environments where they coexist with humans (e.g., as domestic assistants) requires special attention to trustworthiness and explainability. However, the use of LLMs and other deep learning models still does not resolve these key issues. Deep learning systems may hallucinate, be unable to justify their decisions as black boxes, or perform badly on unseen scenarios. In this work, we propose the use of s(CASP), a goal-directed common sense reasoner based on Answer Set Programming, to break down the high-level tasks of an autonomous agent into mid-level instructions while justifying the selection of these instructions. To validate its use in real applications we present a framework that integrates the reasoner into the VirtualHome simulator and compares its accuracy with GPT-4o, running some of the "real" use cases available in the domestic environments of VirtualHome. Additionally, since experiments with VirtualHome have shown the need to reduce the response time (which increases as the agent's decision space grows), we have proposed and evaluated a series of optimizations based on program analysis that exploit the advantages of the top-down execution of s(CASP).

1 Introduction

As autonomous aids, both robotic and digital, have become more commonplace, a need has surfaced for more trustworthy and generalizable task breakdown. Tasks which humans can perform instinctively (e.g., make a sandwich) need to be broken down into composite executable mid-level steps by an autonomous agent (grab bread, grab peanut butter, etc.). To accomplish this breakdown, modern research uses deep learning systems [12] that consume large quantities of data to replicate actions and patterns represented in that data. Large language models (LLMs) in particular have grown in both function and popularity in recent years. However, LLMs and other deep learning-based systems do not reason the same way that humans do and thus are prone to hallucinations and mistakes unlike those that would be made by a human. Even when it is well known that a certain model will produce erroneous output, the models often cannot explain their decisions in a way that allows for correction, leading to new vulnerabilities [17]. This lack of explainability leads to lower trust in the model's decisions [6].

Furthermore, a recent study by the Association for the Advancement of Artificial Intelligence [16] found that 76% of AI professionals surveyed believed that current approaches were "unlikely" or "very unlikely" to yield true artificial general intelligence. Despite these notable downsides in the use of deep learning systems, LLMs have continued to proliferate. Beyond the generation of natural language, researchers are searching for ways to use LLMs in embodied environments. That is an environment, simulated or real, where an autonomous agent is "embodied", that is, able to move around the space and interact with it. Research into using LLMs for the control of robotics is already underway, but continues to struggle with the aforementioned problems as noted in a survey by Wang et al. [20]. Such systems have

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 76–88, doi:10.4204/EPTCS.439.7

© A.R. Tudor, J. Arias & G. Gupta
This work is licensed under the
Creative Commons Attribution License.

already been explored in virtual environments as in the research by Huang et al. [8] with mixed results. We posit that for trustworthy autonomous models, a new approach is required. We propose a symbolic and explainable approach to common sense reasoning using s(CASP), a goal-directed constraint answer set programming system that allows for powerful and justifiable reasoning.

We demonstrate the use of s(CASP) for common sense reasoning on the problem of breaking down high-level tasks into mid-level instructions and compare it to results using LLMs. We will first provide background on relevant prior works in Section 2, including the simulated environment used for validation and the metrics proposed by Huang et al. [8] that we will be using for comparison to the LLM system for task completion. In that section we also discuss s(CASP) and a brief comparison to other planning solutions. Our system, which we call Virtually Embodied Common Sense Reasoning (VECSR), is described in Section 3. To use a symbolic system for reasoning in an embodied environment is a novel approach. We describe our method of modeling real-world information in s(CASP) in Section 3.1, breaking that information into actionable tasks in Section 3.2, and our approach for overcoming the unique challenges associated with the large state in Section 3.3.

We compare the results of VECSR to the research of Huang et al. [8] and to the more modern GPT-4o LLM in Section 4. We use the metrics of correctness and executability proposed by Huang et al. [8], and evaluate the system for generalizability to unseen data and time of execution. Our system creates action plans that are more accurate and executable than those created by LLMs while also providing benefits entirely absent in LLM results, such as context-awareness and justifiability.

There are three main contributions of this paper. Firstly, the creation of VECSR, a novel framework that accepts high-level tasks and uses s(CASP) to plan and execute those tasks in a context-aware fashion in a fully-embodied simulated environment. Secondly, optimization tools based on statically analyzing the program at compile time to improve the execution time and tractability of s(CASP) in real-world environments. And lastly, an evaluation of VECSR's correctness and executability versus GPT-4o, an analysis of VECSR's generalizability, and validation of the optimizations in terms of time of execution.

2 Background and Related Works

2.1 VirtualHome: The Simulation Environment

For demonstration of VECSR we use the VirtualHome simulated environment shown in Figure 1. VirtualHome provides several simulated domestic environments with an API through which one or more agents may take actions and retrieve the state (available at http://virtual-home.org). It comes with common sense knowledge about each object (whether it is grabbable, movable, edible, etc.), information about the current state of the object (turned on, opened or closed, etc.), and its relationship with other objects (on top of, close to, inside of, etc.). The primary advantage of VirtualHome over other simulation engines is that it uses a mid-level control schema. This means that we can give the agent commands like "grab remote" rather than dealing with the details of actual movement that would be more appropriate for a robotic controller. For our research, focused on the common sense reasoning portion of the task-completion process, mid-level control allows for reasoning at the level of the human mind.

The creators of VirtualHome have provided a large human-created dataset of decomposed tasks in the format expected by the simulator. The dataset contains 549 activities, each with one or more human-created breakdowns into mid-level executable units that lead to the completion of those tasks. It is an excellent source of tasks to test various common sense reasoning systems proposed. Note that there are more objects and actions present in the dataset than are in the simulation itself. However, VirtualHome remains an excellent tool for demonstration of common sense reasoning in an embodied environment.

Figure 1: The VirtualHome simulated environment provides a variety of apartment layouts for embodied agents to interact with.

We provide several video examples of tasks completed in the VirtualHome environment at https://drive.google.com/drive/folders/154-KlMgywB1arc9xFQLpAs1K2i9t51j8?usp=sharing.

2.2 Metrics to Evaluate Task Completion

Given a task, our goal is to find a sequence of actions that act as instructions to the autonomous agent to complete that task. Huang et al. [8] have proposed two metrics for evaluating this sequence of actions: correctness and executability. Correctness is the measurement of whether a sequence of actions accomplishes the task. To evaluate this metric, Huang et al. had human participants review the generated action sequences and assess whether they believed the sequence of actions was correct or not. Additionally, they speculate that correctness can be measured by comparing the final state reached by the action sequence to that reached by a human completing that same task. Huang et al. stated that this may not be possible due to a need to keep the initial state constant between tasks. This is a limitation of the LLM, which cannot accept the initial state data except as part of the prompt, which leads to unreliable reproducibility of state. Our methodology is context-aware, ensuring a repeatable initial state. This second method for evaluating correctness is used by VECSR when creating plans.

As defined by Huang et al., a set of instructions is executable if it can be correctly parsed and it satisfies the common sense constraints of the system. For parsing, each instruction must meet the syntactical requirements of the VirtualHome system. Satisfying the common sense constraints of the system means that all pre-conditions of each action are met prior to taking the action. The example used highlights that the agent cannot grab milk out of the fridge if the fridge door is still shut. In addition to physical constraints specified by the VirtualHome system, there are also non-physical constraints. For instance, a food like chicken cannot be safely eaten without being cooked. These secondary constraints are easily ignored by the LLM system because of lack of context-awareness. Therefore, instead of being aware that the only food that is in the house is chicken, it would propose an instruction set that has the agent put “food” on a plate and then eat the “food”. Even when such valuable context is provided in the prompt to the LLM, it is still prone to hallucination of additional state information or forgetting existing information over multiple instructions.

In Section 4.1 we use these metrics, correctness and executability, to compare VECSR to an equivalent LLM reasoner.

2.3 s(CASP) vs. Other Planning Systems

Logic programming approaches have several key advantages over deep learning systems. They are inherently explainable, can produce comparable results (see Gupta et al. [7]) for a specific domain, and are compact and logically sound. We use s(CASP) by Arias et al. [1], a goal-directed constraint-based answer set programming (ASP) system that executes ASP programs in a top-down manner similar to Prolog, eliminating the need for grounding. This allows s(CASP) to reason over incomplete information.

s(CASP) outputs partial answer sets containing only the information needed to successfully execute a query.

Because of the large number of facts and rules associated with even simple real-world environments, being able to create partial answer sets is important from an efficiency viewpoint. Additionally, s(CASP) allows for human-like reasoning with classical negation, negation-as-failure, default rules, and even and odd loops over negation. These features become critical for practical task planning, such as in the example below. Consider the following facts:

```
grabbable(mug).
movable(mug).
grabbable(bowl).
```

Expressing the common sense knowledge that normally things that are grabbable are also movable is difficult to express in classical logic, but easily expressed in ASP and s(CASP). This is accomplished by rules such as `movable(X) :- grabbable(X), not -movable(X)`, which states that `movable(bowl)` holds if `grabbable(bowl)` holds and `-movable(bowl)` is not explicitly asserted.

We view the problem of breaking down a task into a sequence of actions as a planning problem. This approach is similar to other planning-based solutions such as those using PDDL [5] or SOAR [10]. The s(CASP) template used is similar in structure to a PDDL program, with the initial state (derived from the simulated environment, see Section 2.1), a set of possible actions along with preconditions and effects of those actions, and a goal defined in terms of a desired end state. However, reasoning in PDDL is constrained as it only supports conservative reasoning without abducibles (used in the example above) and even loops. ASP provides significantly more flexibility, especially for possible future research, such as inferring generalized knowledge from a specific task.

The SOAR Cognitive Architecture comes with its own programming language, however like other works we seek to leverage its architecture without using the SOAR language [18]. Our architecture uses a similar model as SOAR of keeping a working memory loaded while only selecting relevant items from long-term memory as needed. The use of s(CASP) as a reasoner in our case allows for the strengths of SOAR's design combined with the powerful reasoning mentioned in Section 2.3.

3 Design of VECSR: Virtually Embodied Common Sense Reasoning

The design for VECSR, including the connection to the VirtualHome simulation environment, is illustrated in Figure 2. The resulting flowchart consists of five main stages:

1. Converting the VirtualHome state to s(CASP) facts.
2. Combining the state facts with common sense rules and constraints.
3. Optimizing the newly created program.
4. Solving an appropriate query to create an action sequence for a goal task.
5. Executing the instructions.

After step five, the cycle can start again. This section describes the way we convert the VirtualHome state into s(CASP) facts (Section 3.1), the compile-time optimization performed on the knowledge base (Section 3.3), and how we generate the action plans (Section 3.2). The full code for VECSR is available at https://github.com/Alexandara/vecsr.

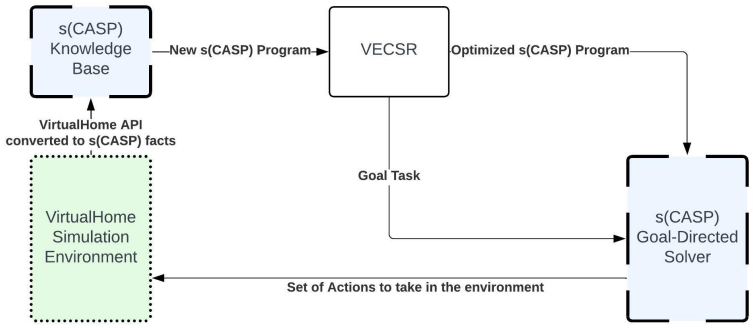


Figure 2: VECSR: framework for s(CASP) common sense reasoning in the VirtualHome simulator.

3.1 Modeling VirtualHome Environments in s(CASP)

A notable weakness of symbolic methods is the representation of large amounts of real-world data for reasoning by a symbolic solver. We have undergone several iterations of s(CASP) templates to represent the simulation environment in a way that is conducive for fast and accurate processing. The VirtualHome API provides data on the environment in the form of a knowledge graph. In this knowledge graph, each object is a node and relationships between objects are edges. In each node there is a variety of state information, which we break into four classes: type, attribute, state, and category. As an example, a specific lamp (lamp1) can have the attribute of having a switch, its state could be off or on, and it could belong to the category of light sources. That information in s(CASP) is:

```
type(lamp1, lamp).
has_switch(lamp1).
on(lamp1).
lightsource(lamp1).
```

We make the simplifying assumption that type, attribute, and category will not change. However, the state of an object could change in the course of an action plan being carried out (in this example, the lamp could be switched off). The same problem occurs with relationships between objects. If the lamp is on a table, `on_top_of(lamp1, table2)`, it can be removed, changing the state. To prevent the old state from clogging up the program during the dynamic part of execution, we use a list to store changeable state:

```
on([lamp1, lamp3]).
on_top_of([[lamp1, table2], [lamp3, floor4]]).
```

This allows the program to maintain the state dynamically while calculating prospective action plans that may accomplish the goal. These facts are then combined with the common sense knowledge base and planning problem rules defined in the next section.

3.2 Breaking Down Tasks into Instructions

We approach the problem of action plan generation as a traditional planning problem. The foundation of this method is that the program starts with an initial state and a final state and attempts to find a plan that satisfies transforming the initial state into the final state. The main loop is as follows:

```
transform(FinalState, Plan) :-
    initial_state(State1), transform(State1, FinalState, [State1], Plan).
transform(State1, FinalState, _, []) :-
    state_subset(FinalState, State1).
transform(State1, FinalState, Visited, [Action|Actions]) :-
    choose_action(Action, State1, FinalState),
    update(Action, State1, State),
    not_member(State, Visited),
    transform(State, FinalState, [State|Visited], Actions).
```

Because the state is described as predicates that are computed dynamically, the initial state is converted into the format expected by the main loop. The initial state rule is in the following format `[ holds([...]), close([...]), on_top_of([...]) | _ ]` where each factor of interest has a list of items that match those parameters. Note that we use lists, `[A, B, C | _]`, where the number of items is defined at run time according to the simulated environment. An example where the agent is carrying a knife and has set a carrot on a cutting board would look like `[ holds([knife1]), close([knife1, carrot2, cuttingboard3, counter4]), on_top_of([[carrot2,cuttingboard3], [cuttingboard3,counter4]]) | _ ]`.[1] Unlike in a traditional planning problem where the actions must lead to the exact final state requested, with VECSR the final state simply needs to be a subset of the state reached by the action plan. For example, if the final state is "holding a phone" then any final state where that is true will be accepted, whether the person is sitting or standing in the final state does not matter.

The actions are selected based on suggestions for the most optimal action to take for any given current and the final state and narrowed down by whether that action is legal or not. If the final state contains the agent sitting on the couch, then the action that completes that goal is to sit on the couch. But the action is not legal unless the agent is standing close to the couch. Legality is defined both by the preconditions of any given action and by the constraints in the knowledge base. So the action that is closest to the final action while still being legal is chosen at every step until the task is complete. The tasks themselves are defined by the final state of the world after the task is complete, meaning that every finite action plan is correct by definition.

We used ten example tasks from the VirtualHome dataset to create the s(CASP) knowledge base. These ten example tasks were chosen based on the first ten example tasks in the paper by Huang et al. [8]. The small number of the sample was due to two factors: the amount of time it took to create a common sense knowledge base for a task and the incredible generalization ability of the s(CASP) program (discussed more in Section 4.2). The planning problem model augmented by dynamic simulated data makes a robust method for action generation based on compact common sense knowledge and reasoning with s(CASP). A small example representing the method for generating action plans is available[2] at `Examples/example_plan.pl` in the repository.

3.3 Optimization With Static Analysis Techniques

When first converting the simulation to s(CASP) facts, we had nearly 3,000 lines of facts by themselves. This was impractical and computationally complex as even the simplest of tasks took over three minutes to accomplish and the more complicated tasks took at least multiple days. For humans, reasoning happens in a matter of seconds, not minutes or hours. To improve the computation time of our system, we used

[1] In this paper we will only display the relevant subset of the state for the example, not the full state as would be kept in the s(CASP) knowledge base.

[2] This file and the rest of examples are linked to https://github.com/Alexandara/vecsr.

three different compile-time optimizations, exploiting the query driven strategy of s(CASP), to prune the knowledge base. The smaller and more relevant the knowledge base, the quicker the processing time.

Modular: The first optimization we implemented was the use of proximity-based "modules" where the s(CASP) knowledge base contains only the state information from relevant areas of the simulation. This kind of analysis works on any data with the concept of "proximity" (such as knowledge graphs or tabulated data) but it works particularly well in an embodied environment where proximity is directly related to physical distance. We segmented the VirtualHome apartment into modules of facts based on rooms such that if the agent was completing a task such as "Brush Teeth" the s(CASP) knowledge base would only include state information for the bathroom.

Dependency Graph: Secondly, to further reduce the knowledge base size, we create a dependency graph for the given query using the knowledge base rules, similar to that proposed by Nguyen et al. [13]. We create this dynamic dependency graph by starting from the query goal as the root node. If goal G_1 calls goal G_2, then a dependency exists between G_1 and G_2. Rules that are untouched during the construction of a dependency graph are not considered during execution of the query.

Partial Grounding: Finally, the most critical optimization is partial grounding of the program. Often, a task is ambiguous enough to be satisfied by multiple objects. If the task includes sitting, where? If the task includes reading, which book? The process of finding what entities the agent is going to use to fulfill a task and creating an action plan to accomplish the task can be separated from each other. First, the program locates the objects necessary for the task. Next, it eliminates any facts that do not pertain to those objects. As an example, if the task is to read then the agent would first select something in the simulation environment to read and then discard the rest. This optimization is similar to the modular optimization, but applied at a lower level taking the semantics of a primitive action (e.g., read) into account. Unlike the grounding done by ASP programs like clingo [4], this optimization uses constraints of the task to bind variables to objects in the state at compile-time, then only pulls the state regarding those objects from VirtualHome.

Fully Optimized: Using these three optimizations together we are able to practically reason over the real-world state in near-real-time. Nevertheless, it is important to note that query-based execution of s(CASP) is what makes it feasible to apply models based on Answer Set Programming to the generation of plans in real-life scenarios.

4 Evaluation and Validation

To validate VECSR executability and correctness, we have conducted three evaluations: (i) comparison of the action plans generated with those generated by GPT-4o (Section 4.1), (ii) correctness of VECSR's plans on unseen data/use cases (Section 4.2), and (iii) run-time comparisons to measure the effect of our compile-time optimizations (Section 4.3). For the first and third evaluation, we use ten tasks listed by Huang et al. [8]. These range in complexity from "Go To Sleep", a simple task where the correct answer is to go lay in the bed, to "Change Sheets in Pillow Case" which requires multiple steps of removing one set of bedding and replacing it with another set. All ten examples are available at https://github.com/Alexandara/vecsr/tree/main/Examples, including the two aforementioned and the following: Browse Internet, Wash Teeth, Brush Teeth, Vacuum, Wash Dirty Dishes, Feed Me, Breakfast, and Read.

Prompt:	**GPT-4o Response:**
Task: Relax on sofa	Step 1: Walk to bedroom (or wherever you sleep).
Step 1: Walk to living room	Step 2: Prepare the sleeping area (e.g., adjust pillows, blankets).
Step 2: Walk to sofa	Step 3: Lie down in bed.
Step 3: Sit in sofa	Step 4: Close your eyes and relax your body.
Task: Go to sleep	Step 5: Breathe deeply and allow yourself to drift off to sleep.

Figure 3: Prompt where the task "relax on sofa" was used as an example for the task "go to sleep".

4.1 Comparison to GPT-4o

LLMs are advancing rapidly, with improved LLMs being revealed multiple times a year (see a recent survey by Patil and Gudivada [15]). Since the paper by Huang et al. [8], a new version of GPT was unveiled that performs better than GPT-3.5 on most standard benchmarks. We ran a similar user study as the one performed by Huang et al. on the newer GPT-4o for a fairer comparison using correctness and execution-time as metrics. However, we do not evaluate the GPT-4o generated plans in terms of executability. The main contribution of the paper by Huang et al. was the use of a secondary LLM translation to create more executable plans, something we did not replicate. Without that additional translation, the executability of LLM-generated plans is very poor and would not make a good comparison metric. The highest executability the modified LLM system by Huang et al. was able to achieve was 78.57%, and that executability rating ignored context-aware executability. Additionally, even on the best performing model, the LLM only made executable and correct plans 35.23% of the time. VECSR plans are 100% executable in real simulated environments and by definition are 100% correct. VECSR tasks are defined by their post-conditions, and action plans are only accepted when the post-conditions of the task are complete. Combined with the strict checking of pre-condition-based legality of every action, this guarantees the final action plans are 100% correct.

To use GPT-4o to generate action plans for tasks, we performed the same kind of zero-shot prompting as Huang et al. where we selected similar tasks from the database, like using the task "relax on sofa" as similar to the task "go to sleep", and prompted the model as in Figure 3. The average length of the GPT-4o generated instructions was 12.3 steps compared to the 9.9 steps in the s(CASP) generated instructions. Huang et al. found that executable instruction sets were often shorter than unexecutable ones, and so the instruction sizes line up with the previous research. To evaluate the correctness of the GPT-4o generated action plans we performed a user study where 12 participants reviewed each of the ten tasks and marked the plans as correct or incorrect. Our results showed that GPT-4o performed similarly or better than the tested LLMs in the paper by Huang et al. and created correct plans in 65.91% of cases. This is lower than the best GPT model used by Huang et al., however this is likely attributed to minor differences in the zero-shot prompting. LLMs are very susceptible to being influenced by the way prompts are done. As an experiment, we asked GPT-4o to provide instructions for the task "go to sleep" without providing the example zero-shot prompt, and received a task plan that was 20 steps and over 300 words in length. Nonetheless, these results indicate that the results found by Huang et al. are still relevant for newer versions of GPT. Table 1 shows more detailed information on the correctness and plan length of each task. Additionally, in the table each task has a link to the solutions generated by each system (the icons to the left of the tasks, first is the VECSR solution and second is the GPT-4o solution), which can be found in the `Examples` folder alongside the programs mentioned in Section 3.3. These examples can also be dynamically generated using the VECSR system.

Huang et al. [8] evaluated the human-created action plans and found they were correct 70.05% of

Table 1: Number of steps generated (VECSR vs. GPT) and GPT's correctness.

	# Steps		Correctness*	1st Wrong Step
	VECSR	GPT	GPT	by GPT
Go To Sleep	3	5	83.33% (±0.39)	
Browse Internet	8	10	*50.00%* (±0.52)	Step 2
Wash Teeth	10	16	*16.67%* (±0.39)	Step 15
Brush Teeth	10	21	91.67% (±0.29)	
Vacuum	5	10	83.33% (±0.39)	
Change Sheets [...]	30	12	91.67% (±0.29)	
Wash Dirty Dishes	12	15	83.33% (±0.39)	
Feed Me	8	10	*33.33%* (±0.49)	Step 2
Breakfast	7	16	*25.00%* (±0.45)	Step 2
Read	6	8	*66.67%* (±0.49)	Unclear
Average	9.9	12.3	65.91%	

(*) Omitted VECSR's correctness because (by design) it is 100%. GPT's correctness is the mean score given by 12 participants (in parenthesis, standard deviation).

the time. From our evaluation of the generated action plans, tasks rated close to or above 70% correct are fairly accurate (for the task "Read", which was scored correct 66.67% of the time, the task was correct enough that we were unable to identify the error). For the GPT-4o created plans we generated that scored lower than 70% correct, we performed an analysis on where the plans stopped being correct. For example, the task which was rated the least correct was "Wash Teeth", which produced the following plan:

1. Walk to bathroom
2. Walk to sink
3. Find faucet
4. Switch on faucet
5. Grab toothbrush
6. Find toothpaste
7. Open toothpaste cap
8. Squeeze toothpaste onto toothbrush
9. Close toothpaste cap
10. Put toothbrush into mouth
11. Brush teeth thoroughly (front, back, sides, and chewing surfaces)
12. Spit out toothpaste residue into sink
13. Rinse toothbrush under faucet
14. Switch off faucet
15. Rinse mouth with water and spit
16. Put toothbrush and toothpaste back in their place

This action plan became incorrect in step 15 because the faucet is turned off by the time the action "Rinse mouth with water and spit" is suggested. Of interest, "Wash Teeth" and "Brush Teeth" are very similar tasks for which we used the same zero-shot prompt (substituting only "Brush" for "Wash" in front of "Teeth"). However, the action plan generated for "Wash Teeth" was only 16.67% correct, while "Brush Teeth" produced an action plan that was rated correct 91.67% of the time.

In most cases, the tasks generated by GPT-4o were longer and more verbose than those generated by VECSR. One notable exception is the "Change Sheets and Pillow Cases" task. This is the most complex

task out of the ten examples, and the VECSR plan takes 18 more steps than the GPT-4o plan. This is due to the executability requirement of the VECSR system. For example, one of the instructions generated by GPT-4o is "Take off the pillowcases from all pillows", which would be multiple steps in the VirtualHome action format ("Walk to bedroom" "Walk to pillowcase 1" "Grab pillowcase 1" "Walk to pillowcase 2" "Grab pillowcase 2").

Additionally, many of the other tasks incorrectly generated by GPT-4o contained a type of error that was commonly made in the human-created VirtualHome dataset, where an item is grabbed before it is "found". This susceptibility to flaws in the data is a hallmark of deep learning. By using our common sense reasoning system that connects directly to an embodied simulation environment, we see 100% accurate results. Even though GPT-4o is a better LLM than the one used by Huang et al. [8], it still leaves much to be desired in terms of correctness. And that is even ignoring the lack of context-awareness that make the LLM plans vague and unexecutable. With an LLM, even knowing that the data used for the zero-shot prompting will lead to incorrect answers, it is difficult or even impossible to correct this issue. The VECSR system is fully justifiable, as a proof of the reasoning behind the answer set for a query can be automatically produced (see Arias, Gupta, and Carro [2] for more information on the justification trees generated by s(CASP)). Thus, when errors are found in VECSR output, the system can easily be adjusted so that the error does not occur again.

Going further, a noted weakness of the prior research is that while LLMs can provide steps for an action, they are not context-aware. The LLM has no access to the apartment when providing its steps, so it does not know if the items it is using to accomplish its goal even exist. In our testing with GPT-4o, it would often default to general types ("food" instead of "salmon") or list possibilities ("breakfast food, such as cereal, bread, eggs, etc."). These results are not necessarily incorrect, but it does limit the executability of those generated instructions. Even the more executable version proposed by Huang et al. [8] does not function with multiple objects of the same kind regardless of if more context was provided. Huang et al. evaluated a set of instructions as executable if the instructions used appropriate terminology (sit, grab <item>, etc.) rather than in larger chunks of more natural language ("Go over and sit on your comfortable chair", "Pick up the <item> if it is close by and available", etc.). Although we compared VECSR output with those results, VECSR directly reasons with the environment, guaranteeing further that the objects being interacted with exist.

Overall, VECSR output is more accurate and executable than that generated by an LLM. VECSR's output is also directly connected to the simulation environment.

4.2 Generalizability

Using s(CASP), VECSR generalizes from a small number of examples to perform well on unseen tasks. This is an improvement over neural networks, which require large quantities of data for decent performance on unseen data. As mentioned above, VECSR was created considering only 10 tasks. This means that the common sense knowledge of all the actions needed for correctly accomplishing these 10 tasks was added to its knowledge base. Once these actions are known, VECSR can perform any other task that can be accomplished using a combination of the actions in its knowledge base. To validate VECSR's ability to generalize from the learned tasks, we randomly selected another 55 tasks from the VirtualHome database and asked VECSR to generate action sequences for them (given the knowledge base of actions corresponding to the original 10 tasks). Of those 55 tasks, 39 tasks (70.91%) were completed without adding any additional rules or constraints to the knowledge base. The results of this evaluation are available at `Examples/unseen_data_results.csv` on the repository. These preliminary results reinforce our claim about the generalization of VECSR.

Table 2: Comparison of run-time execution (in seconds).

	Standard	Modular	Dep.Graph	Part.Ground	Fully Opt.	GPT-4o
Go To Sleep	197.41	2.37	229.95	1.67	**0.54**	1.24
Browse Internet	-	10.45	-	2.19	**0.68**	3.23
Wash Teeth	552.87	2.29	278.73	2.30	**0.78**	1.51
Brush Teeth	414.96	2.27	-	2.75	**0.79**	3.91
Vacuum	391.97	3.88	62.68	1.27	**0.56**	2.16
Change Sheets [...]	-	-	-	52.03	14.16	**5.50**
Wash Dirty Dishes	-	-	-	5.06	4.51	**3.41**
Feed Me	-	201.92	-	4.48	**3.01**	4.25
Breakfast	-	-	-	1.67	**1.47**	3.32
Read	485.45	30.44	557.06	2.46	**0.86**	2.78
Average	408.53	36.23	282.11	7.59	**2.74**	3.13

Note: '-' means timeout after 10 minutes. Average does not take these tasks into account.

There are still gaps in the knowledge base due to actions that are entirely unseen and cannot be generalized. For example, in six of the 16 uncompleted tasks (the 37.5%) VECSR failed because none of the initial 10 tasks require opening something. These tasks include "open door" (which explicitly requires opening the door) and "store meat in freezer" (where the freezer must be opened to store the meat). But these tasks (and any tasks that require opening something) would be solvable, without affecting run-time performance, by adding the action "open" to the knowledge base. We provide the required additions at `Examples/open_rules.pl`. Note that compared to the amount of data consumed for comparable performance by a deep learning system, this generalization capability is very valuable.

4.3 Time of Execution

For any common sense reasoning system to be useful in a practical sense, it has to be able to think on a timescale similar to humans [11]. We implemented several optimizations in VECSR, described in Section 3.3, to reduce the amount of time it takes to generate action plans. We tested the system with no optimizations, with each optimization listed in Section 3.3 individually, and all optimization applied together. Additionally, we timed the generation of plans by GPT-4o. The bottom of Section 3.3 gives more details on the specific optimizations and the online availability of the programs run for these results. While the programs were originally dynamically generated via connection with the VirtualHome simulation environment, they have been statically preserved so the experiment in this section can be recreated exactly. The VECSR results presented in this paper are from a base model 2023 MacBook Pro using version 0.24.01.31 of s(CASP).

Table 2 shows the execution times per example task for unoptimized (Standard) programs, each different optimization, a combination of all three, and the time it took for GPT-4o to generate answers. Even though some simple tasks could be executed without being optimized, even the simplest task still took well over three minutes ("Go To Sleep" at 197.41 seconds) and 50% of the tasks took over ten minutes. The most effective single optimization is partial grounding, which brought the average time for completion down from an average of almost 7 minutes among tasks that were completed at all to 7.59 seconds. However, 7.59 seconds is still too long for the average user, and the most complex task took

nearly a minute to calculate a plan for ("Change Sheets and Pillow Cases" at 52.03 seconds). However, by using all three optimizations, 60% of the tasks execute in under a second. Programs fully optimized by our VECSR system generate a plan 0.39 seconds faster on average than GPT-4o.

This comparison does not include training time, which is much longer for an LLM (several months) than for a logic program. For VECSR, the cost is incurred in developing the knowledge base. Also, an LLM can handle any task, whether done correctly or not, while VECSR can only handle those tasks for which it has acquired common sense knowledge (although with a higher level of accuracy).

5 Conclusion

VECSR represents a practical solution for the use of logic programming in high-fidelity real-world simulations. VECSR generates executable and correct action plans to accomplish high-level tasks and then perform those action plans in an embodied simulation environment. Using s(CASP)'s powerful reasoning capabilities and our novel framework we are able to outperform models that use LLMs as their common sense knowledge repository. With only a few sample tasks, we've created a model that is fast, accurate, and generalizable as a proof of concept of the power of logic programming for reasoning autonomy in embodied environments.

Future work for VECSR can follow two main lines of thought: improvements to the system and further research built upon the VECSR foundation. In the future, more complex reasoning could be integrated, such as using event calculus [19] or reasoning over truly incomplete state information. Additionally, while LLMs are brittle alone there is promising research in pairing them with a symbolic system for improved explainability and reliability [21]; which could be used for a natural language-enabled domestic helper for embodied environments. The ability to pair powerful reasoning with a connection to a simulated environment for embodied agents is a promising and practical framework for future research.

References

[1] Joaqin Arias, Manuel Carro, Elmer Salazar, Kyle Marple & Gopal Gupta (2018): *Constraint Answer Set Programming without Grounding*. *Theory and Practice of Logic Programming* 18(3-4), p. 337–354, doi:10.1017/S1471068418000285.

[2] Joaquín Arias, Gopal Gupta & Manuel Carro (2021): *A Short Tutorial on s(CASP), a Goal-directed Execution of Constraint Answer Set Programs.* In: *ICLP Workshops.*

[3] Joaquín Arias, Mar Moreno-Rebato, Jose A Rodriguez-García & Sascha Ossowski (2024): *Automated legal reasoning with discretion to act using s(LAW).* *Artificial Intelligence and Law* 32(4), pp. 1141–1164, doi:10.1007/s10506-023-09376-5.

[4] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2019): *Multi-shot ASP solving with clingo*. *Theory and Practice of Logic Programming* 19(1), p. 27–82, doi:10.1017/S1471068418000054.

[5] Malik Ghallab, Craig Knoblock, David Wilkins, Anthony Barrett, Dave Christianson, Marc Friedman, Chung Kwok, Keith Golden, Scott Penberthy, David Smith, Ying Sun & Daniel Weld (1998): *PDDL - The Planning Domain Definition Language*.

[6] David Gunning, Eric Vorm, Jennifer Yunyan Wang & Matt Turek (2021): *DARPA's explainable AI (XAI) program: A retrospective*. *Applied AI Letters* 2(4), p. e61, doi:10.1002/ail2.61. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/ail2.61.

[7] Gopal Gupta, Huaduo Wang, Kinjal Basu, Farhad Shakerin, Elmer Salazar, Sarat Chandra Varanasi, Parth Padalkar & Sopam Dasgupta (2023): *Logic-based explainable and incremental machine learning*. *Prolog: The Next 50 Years*, p. 346–358, doi:10.1007/978-3-031-35254-6_28.

[8] Wenlong Huang, Pieter Abbeel, Deepak Pathak & Igor Mordatch (2022): *Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents*. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu & Sivan Sabato, editors: *Proceedings of the 39th International Conference on Machine Learning*, *Proceedings of Machine Learning Research* 162, PMLR, pp. 9118–9147, doi:10.48550/arXiv.2201.07207. Available at https://proceedings.mlr.press/v162/huang22a.html.

[9] Nick Huggett (2002): *Zeno's paradoxes*. doi:10.1093/acprof:oso/9780195379518.003.0002.

[10] John E Laird (2019): *The Soar cognitive architecture*. MIT press.

[11] Doug Lenat & Gary Marcus (2023): *Getting from generative AI to trustworthy AI: What llms might learn from Cyc*. *arXiv preprint arXiv:2308.04445*.

[12] Eduardo F Morales, Rafael Murrieta-Cid, Israel Becerra & Marco A Esquivel-Basaldua (2021): *A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning*. *Intelligent Service Robotics* 14(5), pp. 773–805, doi:10.1007/s11370-021-00398-z.

[13] Manh Thang Nguyen, Jürgen Giesl, Peter Schneider-Kamp & Danny De Schreye (2008): *Termination analysis of logic programs based on dependency graphs*. In: *Logic-Based Program Synthesis and Transformation: 17th International Symposium, LOPSTR 2007, Kongens Lyngby, Denmark, August 23-24, 2007, Revised Selected Papers 17*, Springer, pp. 8–22, doi:10.1007/978-3-540-78769-3_2.

[14] OpenAI, Josh Achiam & et al. (2024): *GPT-4 Technical Report*. doi:10.48550/arXiv.2303.08774. arXiv:2303.08774.

[15] Rajvardhan Patil & Venkat Gudivada (2024): *A review of current trends, techniques, and challenges in large language models (llms)*. *Applied Sciences* 14(5), p. 2074, doi:10.3390/app14052074.

[16] Francesca Rossi & et al. (2025): *AAAI 2025 PRESIDENTIAL PANEL ON THE Future of AI Research*. Available at https://aaai.org/about-aaai/presidential-panel-on-the-future-of-ai-research/.

[17] Joseph Spracklen, Raveen Wijewickrama, AHM Sakib, Anindya Maiti, Bimal Viswanath & Murtuza Jadliwala (2025): *We have a package for you! A comprehensive analysis of package hallucinations by code generating llms*. *arXiv preprint arXiv:2406.10279*, doi:10.48550/arXiv.2406.10279.

[18] Theodore Sumers, Shunyu Yao, Karthik Narasimhan & Thomas Griffiths (2024): *Cognitive Architectures for Language Agents*. *Transactions on Machine Learning Research*, doi:10.48550/arXiv.2309.02427. Available at https://openreview.net/forum?id=1i6ZCvflQJ. Survey Certification.

[19] Ondřej Vašíček, Joaquin Arias, Jan Fiedor, Gopal Gupta, BRENDAL HALL, Bohuslav Křena, Brian Larson, Sarat Chandra Varanasi & Tomáš Vojnar (2024): *Early validation of high-level system requirements with event calculus and answer set programming*. *Theory and Practice of Logic Programming* 24(4), pp. 844–862, doi:10.48550/arXiv.2408.09909.

[20] Jiaqi Wang & et al. (2025): *Large language models for robotics: Opportunities, challenges, and perspectives*. *Journal of Automation and Intelligence* 4(1), pp. 52–64, doi:10.48550. Available at https://www.sciencedirect.com/science/article/pii/S2949855424000613.

[21] Yankai Zeng, Abhiramon Rajasekharan, Kinjal Basu, Huaduo Wang, Joaquín Arias & Gopal Gupta (2024): *A reliable common-sense reasoning socialbot built using llms and goal-directed asp*. *Theory and Practice of Logic Programming* 24(4), pp. 606–627, doi:10.1017/S147106842400022X.

[22] Ming Zhou, Nan Duan, Shujie Liu & Heung-Yeung Shum (2020): *Progress in Neural NLP: Modeling, Learning, and Reasoning*. *Engineering* 6(3), pp. 275–290, doi:10.1016/j.eng.2019.12.014. Available at https://www.sciencedirect.com/science/article/pii/S2095809919304928.

[23] Chenguang Zhu (2021): *Chapter 1 - Introduction to machine reading comprehension*. In Chenguang Zhu, editor: *Machine Reading Comprehension*, Elsevier, pp. 3–26, doi:10.1016/B978-0-323-90118-5.00001-1. Available at https://www.sciencedirect.com/science/article/pii/B9780323901185000011.

Logic Tensor Network-Enhanced Generative Adversarial Network

Nijesh Upreti Vaishak Belle
The University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB, UK
n.upreti@ed.ac.uk vbelle@ed.ac.uk

In this paper, we introduce Logic Tensor Network-Enhanced Generative Adversarial Network (LTN-GAN), a novel framework that enhances Generative Adversarial Networks (GANs) by incorporating Logic Tensor Networks (LTNs) to enforce domain-specific logical constraints during the sample generation process. Although GANs have shown remarkable success in generating realistic data, they often lack mechanisms to incorporate prior knowledge or enforce logical consistency, limiting their applicability in domains requiring rule adherence. LTNs provide a principled way to integrate first-order logic with neural networks, enabling models to reason over and satisfy logical constraints. By combining the strengths of GANs for realistic data synthesis with LTNs for logical reasoning, we gain valuable insights into how logical constraints influence the generative process while improving both the diversity and logical consistency of the generated samples. We evaluate LTN-GAN across multiple datasets, including synthetic datasets (gaussian, grid, rings) and the MNIST dataset, demonstrating that our model significantly outperforms traditional GANs in terms of adherence to predefined logical constraints while maintaining the quality and diversity of generated samples. This work highlights the potential of neuro-symbolic approaches to enhance generative modeling in knowledge-intensive domains.

1 Introduction

Generative Adversarial Networks (GANs) have significantly advanced the field of generative modeling by enabling the creation of highly realistic synthetic data. A typical GAN consists of two neural networks: a generator, which produces synthetic data, and a discriminator, which evaluates whether the generated data is real or fake [6]. Through this adversarial training process, GANs have achieved remarkable success in applications such as image synthesis and data augmentation [6, 29]. However, traditional GANs face a key limitation — they lack the ability to incorporate domain-specific knowledge and logical constraints into the generation process. As a result, they may generate samples that, while visually plausible, violate essential domain rules or logical constraints [29].

To address this limitation, several approaches have been developed to embed logical constraints into GAN architecture. One such approach is Constrained Adversarial Networks (CANs), which introduce a differentiable semantic loss to penalize the generation of invalid samples based on symbolic constraints [17]. This allows end-to-end training that enforces logical rules, enabling the generation of structured objects, such as molecular graphs, that adhere to predefined constraints. Similarly, Constrained Deep Generative Models (C-DGMs) incorporate a constraint layer directly into GAN architectures, such as WGANs and TableGAN, to enforce user-defined constraints expressed as linear inequalities [30]. This ensures that generated samples comply with these constraints while maintaining generation quality.

Other works have explored application-specific constraint enforcement. For instance, [36] incorporates physical laws as soft constraints within GAN training, ensuring that generated data respects

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 89–113, doi:10.4204/EPTCS.439.8

© N. Upreti & V. Belle
This work is licensed under the
Creative Commons Attribution License.

key physical properties, such as conservation laws. In another example, CONGAN introduces user-interactive constraints by mapping semantic similarity constraints into a structured space, enabling users to guide the generation process and ensure the output conforms to high-level specifications [7]. Together, these approaches highlight different strategies for embedding constraints directly into GAN architectures to balance sample quality with rule compliance.

Inspired by these advancements, this paper explores a promising new direction: integrating Logic Tensor Networks (LTNs) with GANs to enhance their ability to incorporate logical constraints. LTNs are a neurosymbolic framework that combines deep learning with first-order logic, enabling neural networks to both learn from data and reason with logical rules [28]. LTNs have been successfully applied to tasks such as semantic image interpretation and knowledge graph completion, demonstrating their ability to embed logical rules directly into the learning process [2, 28].

Building on these foundations, we propose Logic Tensor Network-Enhanced Generative Adversarial Network (LTN-GAN), a novel framework that combines LTNs with GANs to embed logical constraints directly into the data generation process. By incorporating the reasoning capabilities of LTNs, LTN-GAN guides the generator to produce data that is both realistic and consistent with predefined logical rules, addressing a key limitation of traditional GANs. This enables the generation of samples that are not only diverse but also logically coherent, which is particularly important in domains where logical consistency is crucial. We conduct extensive experiments on synthetic datasets as well as the MNIST dataset, demonstrating that LTN-GAN outperforms baseline GANs in both sample diversity and logical adherence. By directly integrating logical reasoning into the generative process, LTN-GAN offers a flexible and scalable way to inject domain-specific knowledge into GANs, improving the reliability and semantic quality of the generated data.

2 Related Works

Recent advances in neuro-symbolic generative modeling have focused on combining the flexibility of deep learning with the structure and interpretability of symbolic logic. These hybrid approaches are especially valuable in domains where generated outputs must adhere to explicit rules, such as physical constraints, combinatorial structures, or logical relationships [15, 16]. In this context, integrating symbolic constraints into generative models—particularly GANs—has emerged as a promising direction for improving output validity and controllability.

Several works have explored constrained GAN frameworks that incorporate constraints or rules directly into the generation process [3, 23, 14]. Constrained Adversarial Networks (CANs) [17] introduce constraint solvers into GAN training, modifying the generator's objective to penalize invalid samples. GenCO [5] embeds combinatorial optimization procedures within generative models, ensuring that each generated output satisfies feasibility constraints. Disjunctive Refinement Layers (DRL) [31] refine outputs by dynamically correcting them to conform to disjunctive logical constraints. While effective, these methods are typically designed for structured or discrete domains, and their formulations are often problem-specific.

In addition to general-purpose constrained GANs, domain-targeted applications have demonstrated the benefits of rule-guided generation. BFGAN [18] incorporates lexical constraints in natural language generation by combining forward and backward generators to maintain fluency and coherence. In scientific modeling, Physically Constrained GANs [8] enforce physical laws to improve the plausibility of Earth system model outputs. Similarly, RCFL-GAN [26] applies resource constraints in federated learning environments to ensure models remain within communication and computation limits. These

approaches show the flexibility of constraint-aware generation, but they rely heavily on domain-specific formulations.

Beyond adversarial training, symbolic reasoning techniques have been integrated into generative models through alternative mechanisms. Posterior regularization methods [10] enforce structured constraints by adding penalty terms during training, encouraging models to produce logically consistent samples. Other strategies encode symbolic knowledge into the latent space using decision diagrams or logic-based encodings to guide sample generation [33]. For sequential data, approaches like [35] learn relational constraints through program synthesis, enforcing structure in outputs such as poetry or music. Although these methods illustrate the value of symbolic constraints, they are not typically built for adversarial training setups.

The emerging field of neurosymbolic generation more explicitly bridges neural learning and symbolic reasoning. Generative Neurosymbolic Machines (GNMs) [11] combine latent neural representations with symbolic layers, enabling logical rule enforcement during generation. In visual domains, AbdGen [25] applies abductive reasoning to refine image generation by aligning latent semantic factors with symbolic abstractions. These techniques improve both control and interpretability, though they are often built for general generative models rather than adversarial frameworks like GANs.

Recent advances in neural-symbolic inference further highlight the potential for logic-based control in generative modeling. For instance, constraint-informed inference methods [4] dynamically apply symbolic corrections during test time to improve generalization and adherence to logical rules. Similarly, logic-injection techniques [34] integrate symbolic constraints directly into neural network architectures, offering new ways to guide learning and output generation.

Despite this growing body of work, the integration of Logic Tensor Networks (LTNs) into GANs remains underexplored. LTNs provide a principled, differentiable framework for enforcing first-order logic constraints, yet their use in adversarial training is limited.

3 Proposed Method: LTN-GAN

LTN-GAN is a hybrid generative framework designed to enhance traditional Generative Adversarial Networks (GANs) by integrating domain-specific logical constraints into the training process. This is achieved through the incorporation of LTNs, enabling the generator to not only produce data that resembles real samples, but also satisfies a set of predefined logical rules that encode domain knowledge. In this section, we describe the architectural components, logic integration mechanism, and adaptive training strategies that characterize LTN-GAN.

3.1 Model Architecture

LTN-GAN follows the standard GAN framework, consisting of a generator G and a discriminator D, where G learns to generate data samples that resemble a target distribution, and D learns to distinguish real samples from generated ones. The generator maps a latent vector $z \sim \mathcal{N}(0, I)$ from $\mathbb{R}^{d_z}$ to a sample

$x \in \mathscr{X}$, while the discriminator outputs a probability $D(x) \in (0,1)$ indicating whether the sample is real.[1]

Network Type and Activation Functions: All experiments use Multilayer Perceptrons (MLPs), which are fully connected feedforward neural networks. These consist of a sequence of linear layers interleaved with non-linear activations, specifically LeakyReLU in all hidden layers. Dropout is applied in the discriminator for regularization. For the 2D datasets (Gaussian, Grid, Ring), both G and D are shallow MLPs operating on 2D vectors. The final activation in G is linear (i.e., no bounding activation such as `tanh`), to avoid introducing geometric bias. For MNIST, the generator outputs a 28×28 image with a final Sigmoid activation to bound pixel values in $[0,1]$, while the discriminator includes a Sigmoid output and an auxiliary classifier head.

Generator Architecture: In the 2D datasets (Gaussian, Grid, Ring), G is a three-layer MLP with 128 hidden units, LeakyReLU activations, and a linear output layer. No final activation is applied in Gaussian and Grid. For **Ring**, the output represents a polar coordinate pair (r,θ), which is converted to Cartesian form $(x,y) = (r\cos\theta, r\sin\theta)$, facilitating ring-structured generation. In **MNIST**, G is conditioned on digit labels via one-hot concatenation with the latent vector. The input passes through three fully connected layers with batch normalization and LeakyReLU, then reshaped to produce a 28×28 grayscale image. A final Sigmoid constrains pixel values to the $[0,1]$ range.

Discriminator Architecture: The discriminator D mirrors the MLP structure with LeakyReLU activations and a final Sigmoid for real/fake prediction. In the MNIST setup, an auxiliary classification head (Softmax over classes) is appended to aid the generator in producing recognizable digits and support logic-based digit constraints.

3.2 Logic Tensor Network (LTN) Component

A Logic Tensor Network (LTN) [28, 2] is a neuro-symbolic framework that combines the representational power of first-order logic with the learning capacity of neural networks. In an LTN, domain knowledge is expressed as a set of weighted logic formulas built from *predicates*, *connectives*, and *quantifiers*, all of which are implemented in a differentiable form so that truth degrees can be optimised jointly with neural parameters.

In the classical LTN setting, each predicate $P_k(\cdot;\phi_k) : \mathscr{X} \to [0,1]$ is a trainable neural function mapping a ground term (e.g., a data sample) to a real-valued *truth degree* in $[0,1]$. Logical formulas are constructed from these predicates via fuzzy logic connectives (differentiable generalisations of $\wedge$, $\vee$, $\neg$, $\Rightarrow$), and are quantified over the available data with fuzzy universal ($\forall$) or existential ($\exists$) operators. The degree to which the knowledge base is satisfied is then differentiable with respect to predicate and model parameters, allowing logical constraints to guide learning.

Adaptation to GAN training. In LTN–GAN, the LTN module $\mathscr{C}$ encodes domain knowledge as a set of weighted formulas $\mathscr{K} = \{(F_i, w_i)\}_{i=1}^{m}$. Predicates are implemented as small neural networks operating directly on generated samples $x_G = G(z)$ (e.g., "is on grid," "is digit 3"), producing truth degrees in $[0,1]$. The generator is penalised when its samples violate these rules, by adding the LTN-derived *logic loss* to the adversarial loss.

[1] *Adversarial losses.* The discriminator is trained using binary cross-entropy loss:

$$L_D = L_D^{\text{real}} + L_D^{\text{fake}}, \quad L_D^{\text{real}} = \mathbb{E}_{x\sim p_{\text{data}}} \ell_{\text{BCE}}(D(x), y_{\text{real}}), \quad L_D^{\text{fake}} = \mathbb{E}_z \ell_{\text{BCE}}(D(G(z)), y_{\text{fake}}),$$

$$L_G^{\text{adv}} = \mathbb{E}_z \ell_{\text{BCE}}(D(G(z)), y_{\text{real}}) \equiv -\mathbb{E}_z \log D(G(z)).$$

We use the non-saturating version of the generator loss for stable training. Label smoothing is applied in some datasets (e.g., using 0.9 for real labels), and the binary cross-entropy is defined as $\ell_{\text{BCE}}(p,y) = -y\log p - (1-y)\log(1-p)$.

Formula construction. Formulas F_i are built from predicates using the following fuzzy connectives:

Conjunction:	$a \wedge b = a \cdot b$	(product t-norm),
Disjunction:	$a \vee b = \max(a,b)$ or $a+b-ab$	(max or probabilistic sum),
Negation:	$\neg a = 1-a$,	
Implication:	$a \Rightarrow b = \max(1-a,b)$	(Reichenbach implication).

Disjunction type is chosen per constraint: max for strict "at least one" semantics, probabilistic sum when partial satisfaction should accumulate.

Quantification. To evaluate the satisfaction of a logical formula F_i over a batch B of generated samples, we use a differentiable aggregation scheme based on the power mean:

$$\mathrm{Sat}_{F_i}(B) = \mathrm{Agg}_{p_i}\left(\{F_i(x) : x \in B\}\right),$$

where Agg_p denotes the *power mean* (or generalized mean) with exponent $p \in \mathbb{R}$. This allows us to softly approximate quantifiers in first-order logic.

For universal quantification ($\forall x$), we use a power mean with a small positive exponent (typically $p = 2$). This makes the aggregation sensitive to low truth values, ensuring that even a few violations significantly reduce the overall satisfaction.

For existential quantification ($\exists x$), we apply a complement-based variant to reward the presence of at least one highly satisfying sample. Specifically, we compute:

$$\exists x\, F(x) \quad \approx \quad 1 - \mathrm{Agg}_p(1 - F(x)),$$

which softly approximates the logical maximum over a set by inverting and aggregating low values. This encourages the generator to produce at least one sample that satisfies the constraint well, even if others do not.

Role in adversarial learning. By incorporating L_{logic} into L_G, the generator is optimised not only to fool the discriminator but also to produce outputs that satisfy all domain rules to a high degree. This tight integration allows the symbolic knowledge in $\mathscr{K}$ to directly influence the generator's gradient updates, guiding it towards logically valid regions of the output space without requiring paired training data for the rules themselves.

3.3 Composite Generator Objective

The generator minimizes a composite objective

$$L_G \;=\; \alpha\, L_G^{\text{adv}} \;+\; \lambda_e\, L_{\text{logic}} \;+\; \beta\, L_{\text{aux}}.$$

Here L_G^{adv} is the non-saturating adversarial loss, L_{logic} penalizes violation of the knowledge base, and L_{aux} is task-specific (e.g., class cross-entropy on MNIST), with (α, β) chosen per experiment.[2]

[2] *Dataset-specific details.* **Gaussian:** two individual-level predicates (range and Gaussian shape) with $\forall$ aggregation; linear schedule $\lambda_e\uparrow$ from 0.05 to 0.30 over the first $K = 80$ epochs; label smoothing $(y_{\text{real}}, y_{\text{fake}}) = (0.9, 0.1)$. **Grid:** "on-grid" and coverage ($\exists$) constraints under identical G/D; BCE everywhere; identical architecture and training hyperparameters across baseline and LTN runs; fixed small noise scale in G to form tight clusters. **Ring:** hierarchical constraints (existence of inner/outer rings, mutual exclusivity, dead-zone avoidance, spatial consistency, balance, precision refinement) with adaptive rule weights; a separate constraint-weight λ_e ramps from 2 to 10. **MNIST:** L_{aux} is class cross-entropy; G also includes a *minimal* template blend (10–20% per-sample weight learned from $[z; y]$) to weakly guide strokes; we use a balanced mixture $\alpha = 0.6$, $\beta = 0.3$ and $\lambda_e = 0.1$ for the LTN term. Labels are unsmoothed in MNIST, smoothed in the 2D Gaussian and grid experiments.

3.4 Logic-Weight Scheduling and Adaptive Rule Weighting

We employ a simple, monotone schedule for the global logic weight:

$$\lambda_e = \begin{cases} \lambda_{\text{start}} + \frac{e}{K}(\lambda_{\text{end}} - \lambda_{\text{start}}), & e \le K, \\ \lambda_{\text{end}}, & e > K, \end{cases}$$

which gradually emphasizes L_{logic} as G learns the data manifold. In the ring setup we additionally adapt per-rule weights from satisfaction feedback. Let $s_{i,t} = \text{Sat}_{F_i}(B_t)$ be the current satisfaction and $w_{i,t}$ the current weight. We update

$$\tilde{w}_{i,t} = \text{clip}\left(w_{i,t} \cdot \begin{cases} 1+\eta, & s_{i,t} < 0.3, \\ 1-\frac{1}{2}\eta, & s_{i,t} > 0.8, \\ 1+\eta\,(0.6 - s_{i,t}), & \text{otherwise}, \end{cases} ; w_{\min}, w_{\max} \right), \qquad w_{i,t+1} = m\, w_{i,t} + (1-m)\,\tilde{w}_{i,t},$$

with momentum m and learning rate η. Here, $\text{clip}(x; a, b) = \min(\max(x, a), b)$ is used in the sense of bounding a value within fixed limits: if $x < a$ it is set to a, and if $x > b$ it is set to b. This keeps per-rule weights from vanishing or growing unbounded, ensuring they remain within the stable range $[w_{\min}, w_{\max}]$. This pushes poorly satisfied rules up and relaxes consistently satisfied ones, stabilizing training while tightening precision late in training. Refer to (Appendix 1) for further algorithmic details.

3.5 Dataset-Specific Predicate Sets

While the overall adversarial–logic framework remains consistent across experiments, the specific set of logical predicates and their aggregation strategies are tailored to each domain. Each predicate $P(\cdot)$ is implemented as a small neural network that outputs a truth degree in $[0, 1]$. Constraints are formulated as first-order fuzzy logic sentences composed using universal ($\forall$) and existential ($\exists$) quantifiers, conjunctions ($\wedge$), disjunctions ($\vee$), and implications ($\Rightarrow$). We realize quantifiers via soft power means, use the product t-norm for conjunction, and apply either maximum or probabilistic sum for disjunction. Negation is implemented as $\neg a = 1 - a$.

To support the effective learning of these predicates—especially in synthetic domains where data geometry is critical—we employ mild geometric initialization strategies across tasks. For instance, we use grid-centered initializations for the GRID dataset, radial priors in RING, and template blending in MNIST. These inductive biases do not enforce the final structure but act as a scaffold for symbolic predicates to operate meaningfully during early training.

Gaussian (2D). This task involves generating samples from a 2D Gaussian-like distribution centered at the origin (Appendix B.1). We define two individual-level predicates evaluated on each generated point. The first, $\text{InRange}(x) = \sigma(3 - \|x\|_\infty)$, softly penalizes points that fall outside a bounded square, where $\|x\|_\infty$ is the maximum coordinate magnitude and σ denotes the sigmoid function. The second, $\text{GaussianShape}(x) = \exp\left(-\frac{1}{2}(\|x\|/2.5)^2\right)$, encourages concentration near the origin by assigning higher scores to points close to the Gaussian mode. Both predicates are enforced via the universal formula:

$$\forall x \left(\text{InRange}(x) \wedge \text{GaussianShape}(x)\right),$$

which guides the generator to produce points that are both spatially bounded and centrally concentrated. Satisfaction is computed as the mean of predicate truth degrees over the batch.

Grid (2D). This task aims to generate samples arranged in a structured 2×2 grid layout, comprising four evenly spaced grid centers (Appendix B.2). The predicate $\text{OnGrid}(x) = f_{\theta_{\text{grid}}}(x)$, implemented via a trainable MLP, evaluates whether a point x lies close to any grid center, and is enforced via the universal constraint $\forall x\ \text{OnGrid}(x)$. To ensure coverage across the grid, we define four cell-specific predicates $\text{InCell}_i(x)$ for each cell $i = 1, \ldots, 4$, and enforce existential constraints $\exists x\ \text{InCell}_i(x)$ to guarantee that each cell contains at least one generated point. The full logical specification is:

$$\Big(\forall x\ \text{OnGrid}(x)\Big) \wedge \bigwedge_{i=1}^{4} \Big(\exists x\ \text{InCell}_i(x)\Big).$$

This encourages the generator to produce samples that are both geometrically regular and evenly distributed across the grid layout.

Ring (2D). This experiment targets a synthetic dataset consisting of two concentric circular regions: an *inner ring* and an *outer ring*, separated by a *dead zone* in between (Appendix B.3). The goal is to generate samples distributed along both rings, avoiding the intermediate region, and satisfying a hierarchy of six logical constraints. The predicates used include: $\text{InnerRing}(x)$ and $\text{OuterRing}(x)$, which identify whether point x lies within the respective annular bands; $\text{DeadZone}(x)$, indicating whether x falls in the undesired area between the rings; and $\text{NearInnerCenter}(x)$ and $\text{NearOuterCenter}(x)$, which promote placement near the ideal radii for each ring.The full constraint set is: (i) *Existence*: both rings must be represented in the sample set, via $\exists x\ \text{InnerRing}(x)$ and $\exists x\ \text{OuterRing}(x)$; (ii) *Mutual exclusivity*: no point should belong to both rings, enforced as $\forall x\ \neg(\text{InnerRing}(x) \wedge \text{OuterRing}(x))$; (iii) *Dead-zone avoidance*: points must not fall in the separating region, via $\forall x\ \neg\text{DeadZone}(x)$; (iv) *Spatial consistency*: ring assignments must reflect true geometric positions; (v) *Balance*: both rings should contain similar proportions of samples, modeled via a soft equality constraint over counts; (vi) *Precision refinement*: points on each ring should also lie near their ideal radius, formalized through implications like $\forall x\ (\text{InnerRing}(x) \Rightarrow \text{NearInnerCenter}(x))$ and similarly for the outer ring. The inner and outer ring tolerances are progressively tightened throughout training to increase precision over time.

MNIST. In the MNIST experiment [13], the knowledge base combines class-specific and structural predicates to ensure both semantic and visual integrity of generated digits (Appendix B.4). For classification, we define one predicate $\text{IsDigit}_k(x)$ per digit class $k \in \{0, \ldots, 9\}$, where each is implemented as a neural classifier that predicts whether an image x corresponds to digit k. To assess visual quality, we introduce a set of general-purpose validity predicates: $\text{ValidPixels}(x)$ encourages pixel intensities to remain within a meaningful dynamic range; $\text{IsConnected}(x)$ promotes topological coherence by checking for a single connected foreground region; $\text{IsComplete}(x)$ ensures that digit shapes are closed or well-formed; and $\text{HasProperIntensity}(x)$ penalizes overly faint or saturated outputs, promoting contrast balance. We enforce class exclusivity via $\forall x\ \bigvee_k \text{IsDigit}_k(x)$ and mutual exclusivity through $\forall x\ \bigwedge_{k \neq m} \neg(\text{IsDigit}_k(x) \wedge \text{IsDigit}_m(x))$. Additional implication rules, such as $\forall x\ \big(\text{ValidPixels}(x) \Rightarrow (\text{IsConnected}(x) \wedge \text{IsComplete}(x))\big)$, guide the generator to produce digits that are not only classifiable but also visually consistent and structurally valid.

3.6 Training Procedure

Across all experiments, training alternates between one discriminator update and one generator update per mini-batch. Let $B \subset \mathscr{D}$ denote a mini-batch of real samples at epoch e.

Discriminator step. We sample real examples $x_R \sim B$ and latent vectors $z \sim \mathcal{N}(0,I)$, generate fake examples $x_G = G(z)$, and *detach* x_G to prevent generator gradients from propagating during the discriminator update. The discriminator is trained with binary cross-entropy (BCE) losses on real and fake samples:

$$L_D^{\text{real}} = \ell_{\text{BCE}}\big(D(x_R), y_{\text{real}}\big), \quad L_D^{\text{fake}} = \ell_{\text{BCE}}\big(D(x_G), y_{\text{fake}}\big),$$
$$L_D = L_D^{\text{real}} + L_D^{\text{fake}}.$$

Here $\ell_{\text{BCE}}(p,y) = -y\log p - (1-y)\log(1-p)$ is computed with probabilities after the sigmoid output. Label smoothing is applied in the 2D datasets (Gaussian, Grid, Ring) with $(y_{\text{real}}, y_{\text{fake}}) = (0.9, 0.1)$, and omitted in MNIST ($(1,0)$ targets). After computing L_D, we backpropagate and update D's parameters.

Logic evaluation. The same batch of generated samples x_G is passed to the LTN $\mathscr{C}$, which evaluates each logical rule $F_i \in \mathscr{K}$, producing satisfaction scores $\text{Sat}_{F_i}(B) \in [0,1]$. Each rule's satisfaction is computed from its predicate truth degrees using the fuzzy connectives and aggregated via its specified quantifier ($\forall$ or $\exists$). These are combined into the global satisfaction

$$S_{\text{logic}}(B) = \frac{\sum_i w_i \, \text{Sat}_{F_i}(B)}{\sum_i w_i},$$

where w_i are fixed or adaptive rule weights depending on the dataset. The logic loss is

$$L_{\text{logic}}(B) = 1 - S_{\text{logic}}(B).$$

In the Ring dataset, w_i are updated adaptively based on satisfaction trends; in other datasets they are fixed.

Generator step. We resample $z \sim \mathcal{N}(0,I)$, regenerate $x_G = G(z)$, and compute the non-saturating adversarial loss:

$$L_G^{\text{adv}} = \ell_{\text{BCE}}\big(D(x_G), y_{\text{real}}\big) \equiv -\mathbb{E}_z \log D(G(z)).$$

For MNIST, we also compute an auxiliary classification loss L_{aux} from the discriminator's class head, comparing predicted labels to the conditioning labels used by the generator. The generator's total loss is

$$L_G = \alpha L_G^{\text{adv}} + \lambda(e) L_{\text{logic}} + \beta L_{\text{aux}},$$

where α and β are dataset-specific coefficients (e.g., $\alpha = 0.6$, $\beta = 0.3$ for MNIST), and $\lambda(e)$ follows the dataset-specific logic-weight schedule from Sec. 3.4. We backpropagate L_G and update G (and any learnable predicate parameters in $\mathscr{C}$, if present).

Logging. During training we record L_G, its adversarial component L_G^{adv}, and L_D; discriminator accuracy (average of real- and fake-sample accuracies); the global logic satisfaction $S_{\text{logic}}(B)$; and the current logic weight $\lambda(e)$. In the Ring dataset, we additionally log each rule's satisfaction and its adaptive weight, while in MNIST we also track the auxiliary classification accuracy.

Training consistency. Optimizer settings are identical between the baseline GAN and LTN–GAN variants: Adam with $(\beta_1, \beta_2) = (0.5, 0.999)$, dataset-specific learning rates, and fixed batch sizes. The update schedule is always one D step followed by one G step. The only differences are in the set of predicates, presence of L_{logic} and L_{aux}, and the $\lambda(e)$ schedule.

3.7 Experimental Setup

We compare LTN–GAN to a baseline GAN identical in generator/discriminator architectures, optimizer settings, label smoothing, and update schedule, differing only in the inclusion of the logical component. Experiments span three synthetic 2D datasets—*Gaussian* (bounded-range and Gaussian-shape predicates), *Grid* (2×2 alignment and cell-coverage constraints), and *Ring* (existence, exclusivity, precision, balance, and dead-zone avoidance)—plus MNIST (28×28 digits with class-consistency and visual-validity rules). The 2D generators are MLPs with LeakyReLU, dropout, and linear $\mathbb{R}^2$ outputs; the MNIST generator maps latent code and class embedding to image space via Sigmoid output. Discriminators are MLPs with LeakyReLU and dropout, with an auxiliary classification head in MNIST. Training horizons follow the code: 100 epochs (Gaussian), 120 (Grid), 150 (Ring), and a 5-step demonstration loop (MNIST), with one D and one G update per batch. All runs use Adam ($\beta_1 = 0.5$, $\beta_2 = 0.999$), BCE losses with label smoothing $(0.9, 0.1)$ in 2D and $(1, 0)$ in MNIST, and an auxiliary classification loss for MNIST. The logic-weight $\lambda(e)$ follows a dataset-specific schedule: linear ramp (Gaussian), fixed constant (Grid), adaptive phase-based with per-rule reweighting (Ring), and fixed small constant (MNIST). Apart from the LTN loss, all hyperparameters, seeds, and training protocols are identical across baseline and LTN–GAN variants. Across datasets, predicates are either fixed analytical functions (e.g., `InRange` and `GaussianShape` in the Gaussian task) or learned neural networks (e.g., `OnGrid`, $\texttt{InCell}_i$, and MNIST digit classifiers). Learned predicates are trained jointly with the generator and discriminator and output soft truth values in $[0, 1]$, enabling gradient-based optimization over logical formulas.

3.8 Evaluation Metrics

We evaluate models using the quantities computed during training: the global logic satisfaction score $S_{\text{logic}}(B)$, obtained by aggregating per-rule truth degrees $\text{Sat}_{F_i}(B)$ via their fuzzy quantifiers and combining them with fixed or adaptive weights w_i; adversarial losses, including the generator's total loss L_G, its adversarial component L_G^{adv}, and the discriminator loss L_D; and discriminator accuracy, measured as the average of real- and fake-sample classification accuracy. The value of the logic-weight $\lambda(e)$ is recorded at each epoch to visualise the effect of the dataset-specific schedule, with per-rule weight trajectories also tracked in the Ring dataset. For all tasks, we plot L_G, L_G^{adv}, L_D, S_{logic}, and $\lambda(e)$ over epochs; in 2D datasets, scatter plots of generated points are compared to target geometries, while for MNIST, generated image grids are inspected to assess visual quality and digit legibility.

4 Results

We evaluate the effectiveness of LTN-GAN by comparing it against baseline GANs across four datasets: **Gaussian**, **Grid**, **Ring**, and **MNIST** [3]. The comparison focuses on three key aspects: adversarial generator loss, sample quality, and logical rule satisfaction. These metrics reflect both traditional generative performance and the model's ability to incorporate symbolic structure. While baseline GANs rely purely on data-driven learning, LTN-GAN integrates domain-specific constraints through logical supervision. This setup allows us to assess how logic-aware training influences geometric structure, sample diversity, and rule consistency in generated outputs.

[3] Code available at: github.com/nuuoe/LTN-GAN/

Comparison Across Datasets

Table 1 presents the performance of LTN-GAN relative to baseline GANs across all four datasets, reporting generator loss (adversarial and total), discriminator loss, sample quality, and logic satisfaction. In every case, LTN-GAN yields higher quality scores while achieving substantial logical rule satisfaction, which is absent in baseline models. On the **Gaussian** dataset, LTN-GAN produces samples with better statistical structure, improving the quality score from 0.183 to 0.470, with a logic satisfaction of 0.916. The total generator loss increases slightly due to logic penalties (0.671 vs. 0.635 adversarial), but training remains stable. For the **Grid** dataset, spatial consistency and grid alignment improve significantly, with the quality score rising from 0.387 to 0.775 and logic satisfaction reaching 0.823. Generator loss again increases moderately (1.031 total vs. 0.441 adversarial), while discriminator loss remains balanced. On the more complex **Ring** dataset, LTN-GAN generates radially balanced samples across inner and outer rings, achieving a strong quality score (0.964 vs. 0.562) and logic satisfaction of 0.817, albeit with higher generator loss due to multiple spatial constraints. Finally, on **MNIST**, logic-aware supervision improves digit structure and recognition, raising the quality score from 0.360 to 0.395 and achieving near-perfect logic satisfaction (0.978), even though discriminator loss is higher due to stronger classification. These results suggest that LTN-GAN effectively balances adversarial and symbolic objectives, improving generative performance across diverse domains.

Dataset	Model	G Loss (Adversarial/Total)	D Loss	Quality Score	Logic Satisfaction
Gaussian	Baseline GAN	0.830	1.390	0.183	N/A
Gaussian	LTN-GAN	0.635/0.671	1.488	**0.470**	**0.916**
Grid	Baseline GAN	0.700	0.692	0.387	N/A
Grid	LTN-GAN	0.441/1.031	0.693	**0.775**	**0.823**
Ring	Baseline GAN	0.708	1.334	0.562	N/A
Ring	LTN-GAN	3.566/3.566	1.364	**0.964**	**0.817**
MNIST	Baseline GAN	1.717	2.319	0.360	N/A
MNIST	LTN-GAN	0.727/0.279	2.789	**0.395**	**0.978**

Table 1: LTN-GAN vs Baseline GAN performance comparison across datasets.

Ablation Studies

We conduct dataset-specific ablation studies to assess how different logic configurations influence sample generation and structural fidelity. Each variant tests the presence or absence of constraint strength, scheduling dynamics, or auxiliary supervision components. Refer to Appendix 6 for detailed descriptions of the domain-specific metrics used to quantify generative quality, structural alignment, and logical satisfaction.

Gaussian Dataset: Table 2 reveals that both statistical and geometric adherence improve substantially under logical constraints. The `ltn_high_constraint` variant achieves the highest combined quality (0.598) with a logic satisfaction of 0.925, confirming that stricter constraints enhance structure. In contrast, baseline and constraint-free variants suffer in both structure and error metrics. The `ltn_no_constraints` variant still benefits from lifted reasoning over soft predicates, even without explicit logical formulas.

Grid Dataset: From Table 3, logic-driven variants show higher clustering precision and complete grid coverage. While `fast_scheduling` underperforms in coverage due to insufficient constraint ex-

Variant	Gaussian Adherence	Statistical Quality	Combined Quality	Mean Error	Std Error	Logic Satisfaction
baseline_gan	0.200	0.165	0.183	0.641	0.175	0.000
full_ltn_gan	0.476	0.465	0.470	0.431	0.719	0.916
ltn_no_constraints	0.402	0.501	0.451	0.327	0.670	0.916
ltn_high_constraint	0.608	0.587	**0.598**	**0.030**	0.673	0.925
ltn_fast_scheduling	0.339	0.637	0.488	0.043	0.527	0.903
ltn_slow_scheduling	0.393	0.408	0.400	0.946	0.508	0.849

Table 2: Gaussian dataset ablation: logic scheduling and logical constraints improve statistical structure.

posure, it maintains high sample quality. The baseline fails to align samples to target regions, resulting in only 607 out of 1000 being placed within the defined grid boundaries. In contrast, all logic variants ensure full alignment through existential grid-cell constraints.

Variant	Grid Cluster	Coverage	Quality Score	Overall Performance	In Targets	Logic Satisfaction
baseline_gan	0.607	0.745	0.387	0.573	607	0.000
full_ltn_gan	1.000	0.638	0.775	**0.882**	1000	0.823
ltn_no_constraints	0.057	0.718	0.205	0.288	57	0.500
ltn_high_constraint	1.000	0.591	0.793	0.877	1000	0.809
ltn_fast_scheduling	1.000	0.179	0.778	0.790	1000	0.802
ltn_slow_scheduling	1.000	0.445	0.721	0.830	1000	0.806

Table 3: Grid dataset ablation: logic constraints and cluster adherence.

Ring Dataset: As shown in Table 4, LTN variants significantly outperform the baseline in ring adherence, balance, and dead zone avoidance. Removing progressive training phases or hierarchical rule weights slightly reduces performance but still maintains high logical satisfaction. Notably, the `simple_constraints` configuration matches or surpasses more complex setups in balance and rule compliance, suggesting that even a flattened constraint structure can benefit from the symbolic inductive bias.

Variant	Ring Adherence	Inner	Outer	Balance	Dead Zone Avoidance	Logic Satisfaction
baseline_gan	0.524	231	0	0.000	0.462	0.000
full_ltn_gan	0.970	291	188	0.785	0.958	0.817
no_constraints	0.462	76	156	0.655	0.868	0.702
no_hierarchical_weights	0.970	265	217	0.900	0.964	0.826
no_progressive_phases	0.922	258	209	0.895	0.934	**0.856**
simple_constraints	0.960	258	224	**0.929**	0.964	0.824

Table 4: Ring dataset ablation: symmetry, balance, and logical consistency.

MNIST Dataset: Table 5 shows that logic-based models achieve better digit recognition and structure, particularly when template priors are included. Removing templates leads to reduced recognition and logic satisfaction, demonstrating their importance for early guidance. Interestingly, even the `weak_ltn` configuration matches or exceeds other variants in digit classification, highlighting the robustness of symbolic supervision when paired with data-level diversity.

Variant	Quality Score	Coverage	Digit Recognition	Template Dependence	Logic Satisfaction
baseline_gan	0.360	0.069	0.717	0.700	0.000
full_ltn_gan	0.395	0.070	0.725	0.700	**0.978**
no_ltn_constraints	0.392	0.069	0.720	0.700	0.000
no_templates	0.367	0.057	0.685	0.000	0.977
weak_ltn	0.391	0.069	**0.731**	0.700	0.977
strong_ltn	**0.398**	0.069	0.727	0.700	0.978

Table 5: MNIST ablation study: logic boosts recognition and semantic accuracy.

4.1 Understanding Output Variability and the Impact of Logic Design

Our ablation studies across all datasets reveal several important factors that influence the variability observed in LTN-GAN's performance. While the integration of logic clearly improves structure, semantics, and generalization, the degree of improvement depends significantly on how the logical components are designed and implemented.

One key source of variability comes from how the logical constraints are formulated. For example, in the Ring dataset, we find that using overly simple constraints results in poorer geometric structure and reduced balance across sample categories. On the other hand, hierarchical and semantically rich logic leads to better ring adherence and improved logical satisfaction. This demonstrates that meaningful constraint design is essential for generating structured samples, especially in tasks where geometry or symbolic relationships matter.

Another important factor is the way logic is applied over the course of training. Variants that use progressive scheduling — gradually increasing the influence of logic over time — consistently outperform those where constraints are applied uniformly or abruptly. This staged approach allows the model to first explore the data space and later refine outputs according to logical structure. The benefit of such curriculum-like training is especially clear in the Ring and Grid datasets.

Constraint weighting also plays a major role. When all constraints are treated equally or fixed in importance, the model tends to either ignore weaker constraints or overfit to strong ones. Our experiments show that dynamic, hierarchical weighting — where constraints are emphasized based on their current satisfaction levels — helps balance between learning general patterns and maintaining logical structure. This strategy leads to better outcomes in both quality and logic adherence.

Additionally, we observe that different datasets respond differently to the same architectural changes. MNIST, for instance, shows stable performance across most ablations, likely due to its well-defined visual structure. In contrast, the synthetic datasets, which require explicit spatial relationships, are much more sensitive to constraint design and scheduling. This highlights the importance of dataset-specific logic design when applying LTN-based generative models.

Finally, our findings confirm a trade-off between sample quality and logic satisfaction. Increasing logical enforcement often leads to reduced sample variability, particularly in strongly constrained variants. This suggests that logic must be carefully balanced with generative flexibility to avoid mode collapse or overly rigid outputs.While LTN-GAN offers significant advantages over traditional GANs, achieving the highest performance requires careful tuning of constraint structure, logic weight scheduling, and dataset-specific considerations. Our results suggest that these models benefit from adaptive strategies rather than fixed configurations, particularly in domains where structural accuracy and semantic validity are essential.

5 Discussion

Our study demonstrates that LTN-GAN effectively integrates symbolic reasoning into the generative process, enabling the generator to produce outputs that not only resemble real data but also conform to domain-specific logical constraints. Across experiments, the model consistently outperformed unconstrained baselines in logic satisfaction and structural quality. However, this integration introduces a well-known tension: enforcing logical consistency often restricts sample diversity, limiting the generator's ability to explore the full data manifold. This trade-off was particularly evident in ablation studies, where higher constraint weights improved logical adherence but occasionally reduced visual variability or coverage. The generator's objective blends adversarial and logical losses using tunable weights α, $\lambda(e)$, and β. These hyperparameters play a critical role in balancing quality, diversity, and rule satisfaction. While our scheduling strategies and architecture-specific adjustments allowed reasonable control, the need for careful tuning indicates that LTN-GAN still lacks adaptive mechanisms for managing this trade-off autonomously.

To situate LTN-GAN within the broader neuro-symbolic landscape, we note that several models embed symbolic structures in latent space to enforce rules indirectly. For example, VAEL [21] incorporates logic programming into VAEs, while MultiplexNet [9] models hierarchical categorical variables to guide generation. These approaches, like LTN-GAN, aim to impose structure, but differ by relying on probabilistic embeddings rather than direct logic reasoning within training objectives.

Other strategies approximate symbolic constraints with differentiable surrogates, making them compatible with gradient-based learning. Semantic Loss [32] was one of the first to express logical rules as differentiable penalties. This has inspired methods like Pseudo-Semantic Loss [1], which localizes constraint enforcement in autoregressive models, and Semantic Objective Functions [20], which embed logic into loss functions using tools from information geometry. These approaches are powerful but often not tightly coupled with adversarial dynamics, where unstable gradients and non-stationary objectives complicate constraint enforcement.

Post-hoc constraint satisfaction through projection or refinement is another promising direction. For instance, SketchGen [24] and Vitruvion [27] enforce geometric consistency in CAD sketches, while Langevin-based refinement [12] iteratively nudges generated outputs toward feasible regions. These methods can boost constraint satisfaction without affecting the core generative process, but they sacrifice training-time reasoning and may result in inefficiencies or brittle outputs.

Our findings also highlight the limitations of hard vs. soft constraint enforcement. While hard constraints ensure compliance, they can restrict diversity and exacerbate mode collapse. In contrast, soft penalties offer flexibility but often lack guarantees. Hybrid mechanisms have emerged to bridge this gap—e.g., Straight-Through Estimators for discrete constraints [34], and dual optimization methods that balance feasibility and exploration [5, 19]. LTN-GAN adopts a soft logic-loss formulation, but further refinement may be needed to avoid local minima or underutilized rules.

One of the central gaps in current neuro-symbolic GANs is the lack of deep integration between symbolic reasoning and adversarial learning. Most models inject logic indirectly—through latent space priors or post-hoc corrections—rather than shaping generator updates directly. For instance, works like [25] rely on rule-injected representations but do not adapt rules dynamically. LTN-GAN takes a step forward by making logic satisfaction differentiable and jointly trainable, but it still relies on predefined rule sets. Future extensions may benefit from dynamic rule induction, where constraints are learned from data or inferred contextually. Meta-abduction [25] and latent-symbolic alignment [4] offer promising foundations for this direction, while advances in goal-directed generation of structured discrete data [22] demonstrate complementary strategies for enforcing properties directly through learned reward op-

timization.

Scalability also remains an open challenge. While LTN-GAN performs well on structured datasets like MNIST and synthetic geometries, scaling to higher-dimensional tasks (e.g., natural images or molecular graphs) is non-trivial. One direction involves hierarchical or geometric solvers [27], which may enable logic to operate at multiple levels of abstraction. Additionally, integrating constraint-aware discriminators—capable of assessing rule violations during training—could improve logic enforcement without sacrificing diversity.

Overall, our work contributes to the emerging field of constraint-aware generative modeling, aligning with efforts to combine logic and learning [17, 30, 33, 10]. Recent frameworks such as Generative Neurosymbolic Machines [11] and abductive refinement methods [25] reinforce the potential of symbolic reasoning to enhance generative control. However, robust integration of logic within adversarial training remains underdeveloped. Addressing key challenges in constraint optimization, adversarial stability, and dynamic logic learning will be essential for realizing the full potential of neuro-symbolic GANs.

5.1 Broader Implications for Generative Reasoning

While the current study evaluates LTN-GAN within the standard generative adversarial framework, it is important to acknowledge that the potential of Logic Tensor Networks extends beyond improving sample quality or diversity in such settings. The constrained and adversarial nature of GANs often limits the full expressive power of symbolic logic integration. In particular, the generator is still primarily optimized for adversarial objectives, which may conflict with deeper forms of structured reasoning.

We believe that the true strength of LTNs lies in their ability to support logic-guided generative reasoning. This is especially relevant for tasks like scene generation, compositional image synthesis, or multi-modal symbolic grounding—areas where the generation process involves satisfying complex relationships, spatial configurations, and rule-based dependencies. In such contexts, LTNs can act not just as a regularizer but as a driver of structured generation, enforcing high-level semantic rules during the generative process itself. Future work should explore this direction by integrating LTNs into architectures beyond GANs, such as diffusion models, autoregressive decoders, or neuro-symbolic scene simulators, where symbolic reasoning can directly shape content generation rather than merely constrain it post hoc.

6 Conclusion

LTN-GAN represents a step forward in neuro-symbolic generative modeling, demonstrating that logical constraints can effectively guide adversarial learning. However, challenges such as the diversity-adherence trade-off, manual rule specification, and scalability constraints highlight key areas for future improvement. Research on adaptive rule learning, constraint-aware adversarial training, and scalable logic enforcement techniques will be crucial for advancing neuro-symbolic GANs into more robust and flexible frameworks. Addressing these challenges will enable GANs to generate logically structured, high-quality outputs across diverse domains while retaining generative diversity and flexibility.

7 Acknowledgments

Vaishak Belle was funded by a CISCO grant and a Royal Society University Research Fellowship.

References

[1] Kareem Ahmed, Kai-Wei Chang & Guy Van den Broeck (2023): *A Pseudo-Semantic Loss for Autoregressive Models with Logical Constraints*. In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. URL http://papers.nips.cc/paper_files/paper/2023/hash/3accfe8332366a6f740d8740cd4cd653-Abstract-Conference.html.

[2] Samy Badreddine, Artur S. d'Avila Garcez, Luciano Serafini & Michael Spranger (2022): *Logic Tensor Networks*. *Artif. Intell.* 303, p. 103649. URL https://doi.org/10.1016/j.artint.2021.103649.

[3] Xiaopeng Chao, Jiangzhong Cao, Yuqin Lu, Qingyun Dai & Shangsong Liang (2021): *Constrained Generative Adversarial Networks*. *IEEE Access* 9, pp. 19208–19218. URL https://doi.org/10.1109/ACCESS.2021.3054822.

[4] Cristina Cornelio, Jan Stuehmer, Shell Xu Hu & Timothy M. Hospedales (2023): *Learning where and when to reason in neuro-symbolic inference*. In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, OpenReview.net. URL https://openreview.net/forum?id=en9V5F8PR-.

[5] Aaron M. Ferber, Arman Zharmagambetov, Taoan Huang, Bistra Dilkina & Yuandong Tian (2024): *GenCO: Generating Diverse Designs with Combinatorial Constraints*. In: *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. URL https://openreview.net/forum?id=DiyE6OOGBa.

[6] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville & Yoshua Bengio (2014): *Generative Adversarial Nets*. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 2672–2680. URL https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html.

[7] Eric Heim (2019): *Constrained Generative Adversarial Networks for Interactive Image Generation*. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, Computer Vision Foundation / IEEE, pp. 10753–10761. URL https://doi.org/10.1109/CVPR.2019.01101.

[8] Philipp Hess, Markus Drüke, Stefan Petri, Felix M. Strnad & Niklas Boers (2022): *Physically constrained generative adversarial networks for improving precipitation fields from Earth system models*. *Nat. Mac. Intell.* 4(10), pp. 828–839. URL https://doi.org/10.1038/s42256-022-00540-1.

[9] Nick Hoernle, Rafael-Michael Karampatsis, Vaishak Belle & Kobi Gal (2022): *MultiplexNet: Towards Fully Satisfied Logical Constraints in Neural Networks*. In: *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pp. 5700–5709. URL https://doi.org/10.1609/aaai.v36i5.20512.

[10] Zhiting Hu, Zichao Yang, Ruslan Salakhutdinov, Lianhui Qin, Xiaodan Liang, Haoye Dong & Eric P. Xing (2018): *Deep Generative Models with Learnable Knowledge Constraints*. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 10522–10533. URL https://proceedings.neurips.cc/paper/2018/hash/d7e77c835af3d2a803c1cf28d60575bc-Abstract.html.

[11] Jindong Jiang & Sungjin Ahn (2020): *Generative Neurosymbolic Machines*. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. URL https://proceedings.neurips.cc/paper/2020/hash/94c28dcfc97557df0df6d1f7222fc384-Abstract.html.

[12] Sachin Kumar, Biswajit Paria & Yulia Tsvetkov (2022): *Gradient-based Constrained Sampling from Language Models*. In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 2251–2277. URL https://doi.org/10.48550/arXiv.2205.12558.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio & Patrick Haffner (1998): *Gradient-based learning applied to document recognition*. *Proc. IEEE* 86(11), pp. 2278–2324. URL https://doi.org/10.1109/5.726791.

[14] Wanxin Li (2020): *Supporting Database Constraints in Synthetic Data Generation based on Generative Adversarial Networks*. In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pp. 2875–2877. URL https://doi.org/10.1145/3318464.3384414.

[15] Zenan Li, Yunpeng Huang, Zhaoyu Li, Yuan Yao, Jingwei Xu, Taolue Chen, Xiaoxing Ma & Jian Lu (2023): *Neuro-symbolic Learning Yielding Logical Constraints*. In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*. URL http://papers.nips.cc/paper_files/paper/2023/hash/4459c3c143db74ee52afebdf56836375-Abstract-Conference.html.

[16] Zenan Li, Zehua Liu, Yuan Yao, Jingwei Xu, Taolue Chen, Xiaoxing Ma & Jian Lü (2023): *Learning with Logical Constraints but without Shortcut Satisfaction*. In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. URL https://openreview.net/forum?id=M2unceRvqhh.

[17] Luca Di Liello, Pierfrancesco Ardino, Jacopo Gobbi, Paolo Morettin, Stefano Teso & Andrea Passerini (2020): *Efficient Generation of Structured Objects with Constrained Adversarial Networks*. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. URL https://proceedings.neurips.cc/paper/2020/hash/a87c11b9100c608b7f8e98cfa316ff7b-Abstract.html.

[18] Dayiheng Liu, Jie Fu, Qian Qu & Jiancheng Lv (2019): *BFGAN: Backward and Forward Generative Adversarial Networks for Lexically Constrained Sentence Generation*. *IEEE ACM Trans. Audio Speech Lang. Process.* 27(12), pp. 2350–2361. URL https://doi.org/10.1109/TASLP.2019.2943018.

[19] Xianggen Liu, Qiang Liu, Sen Song & Jian Peng (2020): *A Chance-Constrained Generative Framework for Sequence Optimization*. In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pp. 6271–6281. URL http://proceedings.mlr.press/v119/liu20i.html.

[20] Miguel Ángel Méndez Lucero, Enrique Bojorquez Gallardo & Vaishak Belle (2025): *Semantic Objective Functions: A Distribution-Aware Method for Adding Logical Constraints in Deep Learning*. In: *Proceedings of the 17th International Conference on Agents and Artificial Intelligence, ICAART 2025 - Volume 3, Porto, Portugal, February 23-25, 2025*, pp. 909–917. URL https://doi.org/10.5220/0013229200003890.

[21] Eleonora Misino, Giuseppe Marra & Emanuele Sansone (2022): *VAEL: Bridging Variational Autoencoders and Probabilistic Logic Programming*. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*. URL http://papers.nips.cc/paper_files/paper/2022/hash/1e38b2a0b77541b14a3315c99697b835-Abstract-Conference.html.

[22] Amina Mollaysa, Brooks Paige & Alexandros Kalousis (2020): *Goal-directed Generation of Discrete Structures with Conditional Generative Models*. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. URL https://proceedings.neurips.cc/paper/2020/hash/f9b9f0fef2274a6b7009b5d52f44a3b6-Abstract.html.

[23] Paolo Morettin, Andrea Passerini & Stefano Teso (2021): *Co-creating Platformer Levels with Constrained Adversarial Networks*. In: *Joint Proceedings of the ACM IUI 2021 Workshops co-located with 26th ACM Conference on Intelligent User Interfaces (ACM IUI 2021), College Station, United States, April 13-17, 2021*. URL https://ceur-ws.org/Vol-2903/IUI21WS-HAIGEN-9.pdf.

[24] Wamiq Reyaz Para, Shariq Farooq Bhat, Paul Guerrero, Tom Kelly, Niloy J. Mitra, Leonidas J. Guibas & Peter Wonka (2021): *SketchGen: Generating Constrained CAD Sketches*. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 5077–5088. URL https://proceedings.neurips.cc/paper/2021/hash/28891cb4ab421830acc36b1f5fd6c91e-Abstract.html.

[25] Yifei Peng, Yu Jin, Zhexu Luo, Yao-Xiang Ding, Wang-Zhou Dai, Zhong Ren & Kun Zhou (2023): *Generating by Understanding: Neural Visual Generation with Logical Symbol Groundings*. *CoRR* abs/2310.17451. URL https://doi.org/10.48550/arXiv.2310.17451.

[26] Yuyan Quan, Songtao Guo & Dewen Qiao (2024): *RCFL-GAN: Resource-Constrained Federated Learning with Generative Adversarial Networks*. In: *27th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2024, Tianjin, China, May 8-10, 2024*, IEEE, pp. 513–518. URL https://doi.org/10.1109/CSCWD61410.2024.10580839.

[27] Ari Seff, Wenda Zhou, Nick Richardson & Ryan P. Adams (2022): *Vitruvion: A Generative Model of Parametric CAD Sketches*. In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, OpenReview.net. URL https://openreview.net/forum?id=Ow1C7s3UcY.

[28] Luciano Serafini & Artur S. d'Avila Garcez (2016): *Learning and Reasoning with Logic Tensor Networks*. In: *AI*IA 2016: Advances in Artificial Intelligence - XVth International Conference of the Italian Association for Artificial Intelligence, Genova, Italy, November 29 - December 1, 2016, Proceedings*, *Lecture Notes in Computer Science* 10037, Springer, pp. 334–348. URL https://doi.org/10.1007/978-3-319-49130-1_25.

[29] Preeti Sharma, Manoj Kumar, Hitesh Kumar Sharma & Soly Mathew Biju (2024): *Generative adversarial networks (GANs): Introduction, Taxonomy, Variants, Limitations, and Applications*. *Multim. Tools Appl.* 83(41), pp. 88811–88858. URL https://doi.org/10.1007/s11042-024-18767-y.

[30] Mihaela C. Stoian, Salijona Dyrmishi, Maxime Cordy, Thomas Lukasiewicz & Eleonora Giunchiglia (2024): *How Realistic Is Your Synthetic Data? Constraining Deep Generative Models for Tabular Data*. In: *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, OpenReview.net. URL https://openreview.net/forum?id=tBROYsEz9G.

[31] Mihaela C. Stoian & Eleonora Giunchiglia (2025): *Beyond the convexity assumption: Realistic tabular data generation under quantifier-free real linear constraints*. In: *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*, OpenReview.net. URL https://openreview.net/forum?id=rx0TCew0Lj.

[32] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang & Guy Van den Broeck (2018): *A Semantic Loss Function for Deep Learning with Symbolic Knowledge*. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 5498–5507. URL http://proceedings.mlr.press/v80/xu18h.html.

[33] Yexiang Xue & Willem-Jan van Hoeve (2019): *Embedding Decision Diagrams into Generative Adversarial Networks*. In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings*, *Lecture Notes in Computer Science* 11494, Springer, pp. 616–632. URL https://doi.org/10.1007/978-3-030-19212-9_41.

[34] Zhun Yang, Joohyung Lee & Chiyoun Park (2022): *Injecting Logical Constraints into Neural Networks via Straight-Through Estimators*. In: *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, pp. 25096–25122. URL https://proceedings.mlr.press/v162/yang22h.html.

[35] Halley Young, Maxwell Du & Osbert Bastani (2022): *Neurosymbolic Deep Generative Models for Sequence Data with Relational Constraints*. In: *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA,*

USA, November 28 - December 9, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/f13ceb1b94145aad0e54186373cc86d7-Abstract-Conference.html.

[36] Yang Zeng, Jin-Long Wu & Heng Xiao (2021): *Enforcing Imprecise Constraints on Generative Adversarial Networks for Emulating Physical Systems*. *Communications in Computational Physics* 30(3), pp. 635–665. URL https://doi.org/10.48550/arXiv.1911.06671.

A Extended Methodology

A.1 Detailed Experimental Setup

We evaluate LTN-GAN against a baseline GAN that shares the same generator and discriminator architectures, optimization settings, and training schedule, but omits any logic-based supervision. This parallel setup ensures that observed performance differences stem from the integration of logical constraints, rather than incidental differences in architecture or learning dynamics.

Experiments are conducted across three synthetic datasets—*Gaussian*, *Grid*, and *Rings*—as well as the image-based *MNIST* dataset. In all cases, the generator is a fully connected multilayer perceptron (MLP) with LeakyReLU activations and hidden layers of size 128. For the synthetic datasets, the generator receives a 2D latent vector from a standard normal distribution and produces 2D samples with no output activation, enabling unbounded outputs in $\mathbb{R}^2$. The MNIST generator is conditioned on a one-hot digit label concatenated with 100-dimensional Gaussian noise and outputs 28×28 grayscale images. The discriminator matches the generator in structure, outputting a single probability score, and, for MNIST, an auxiliary classification head is added for digit prediction.

Training is performed using the Adam optimizer with learning rate 0.001, $\beta_1 = 0.5$, and $\beta_2 = 0.999$, over 100 epochs. A batch size of 32 is used for synthetic data and 64 for MNIST. Discriminator label smoothing is employed: real samples are labeled with 0.9, and generated samples with 0.1. In all LTN-GAN settings, the generator loss combines adversarial and logical components: $L_G = \alpha L^{\text{adv}} + \lambda(e) L^{\text{logic}} + \beta L^{\text{aux}}$, with $\alpha = 1$, $\beta = 0.2$ (if applicable), and $\lambda(e)$ controlled by a linear ramp-up schedule over the first K epochs. For the Gaussian dataset, λ rises from 0.05 to 0.3 over 80 epochs; for Rings and Grid, from 0.1 to 0.5 over 50 epochs; and for MNIST, from 0.05 to 0.25 over 60 epochs.

Additional stabilizing mechanisms are used in logic-augmented training. For Rings and MNIST, a checkpoint-based backtracking mechanism allows the generator to roll back to one of the previous 20 saved states if a sudden drop in rule satisfaction is detected. MNIST further incorporates a logic-aware curriculum: early training mixes in digit template priors for structure guidance, which are gradually faded out, letting the generator learn symbolic constraints autonomously. In all synthetic datasets, the generator receives added Gaussian noise (standard deviation 0.1) in the latent space to promote exploratory diversity. Evaluation is performed every 10 epochs by sampling 1,000 points from the generator and computing two primary metrics: (i) logical rule satisfaction using fuzzy logic aggregation over the constraint set, and (ii) geometric diversity, measured by pairwise Euclidean distances (or pixel variance for MNIST). MNIST additionally tracks classification accuracy and brightness/centering statistics through learned predicate networks. All experiments are implemented in PyTorch and executed on a MacBook Pro (2.4 GHz 8-Core Intel Core i9, 64 GB RAM, AMD Radeon Pro 5500M 8 GB GPU).

To facilitate symbolic supervision, each dataset uses a custom set of logical predicates tailored to its geometric or semantic structure. For the synthetic domains, these predicates are implemented as fixed mathematical functions or simple threshold-based heuristics over coordinate space. In the Grid and Rings datasets, existential and universal constraints ensure spatial coverage and separation of geometric regions, with satisfaction computed via fuzzy logic aggregation. MNIST predicates, in contrast, are learned via small MLPs and are trained in tandem with the generator to evaluate properties such as digit class, visual clarity, and spatial alignment. Quantifiers are realized using differentiable power means—min-like for universal rules and max-like for existential ones—allowing gradients to flow through logical formulas. Logic satisfaction is tracked per rule, and adaptive weights are optionally applied to focus learning on underperforming constraints. This modular logic evaluation framework enables LTN-GAN to flexibly incorporate symbolic rules without disrupting the adversarial learning loop.

Algorithm 1 LTN–GAN Training Algorithm[1][2]

Require: Training data $\mathscr{D}$, Generator G, Discriminator D, Constraint system $\mathscr{C}$ with rules R, config θ (incl. $\lambda(e)$), optional auxiliary loss $\mathscr{A}$
Ensure: Trained models G^*, D^*

Initialize parameters of G, D, and optionally $\mathscr{C}$
Create optimizers: opt_G, opt_D (and $\text{opt}_{\mathscr{C}}$ if applicable)
Set label smoothing values: $y_{\text{real}} \leftarrow 0.9$, $y_{\text{fake}} \leftarrow 0.1$
Define logic loss weight schedule $\lambda(e)$
for $e = 1$ **to** N_{epochs} **do**
  $\lambda \leftarrow \lambda(e)$
  for batch $B \subset \mathscr{D}$ **do**
    Sample real data $x_R \sim B$, latent noise $z \sim \mathscr{N}(0, I)$
    Generate fake data $x_G \leftarrow G(z)$
    ▷ — Discriminator Update —
    opt_D.zero_grad()
    $L_D^{\text{real}} \leftarrow \ell_{\text{BCE}}(D(x_R), y_{\text{real}})$
    $L_D^{\text{fake}} \leftarrow \ell_{\text{BCE}}(D(\text{detach}(x_G)), y_{\text{fake}})$
    $L_D \leftarrow L_D^{\text{real}} + L_D^{\text{fake}}$
    backprop L_D, opt_D.step()
    ▷ — Logic Constraint Evaluation —
    cons $\leftarrow \mathscr{C}$.evaluate(x_G, R)
    $(S_{\text{logic}}, L_{\text{constraint}}) \leftarrow \mathscr{C}$.aggregate(cons)
    $\mathscr{C}$.update_weights_or_phase(cons)
    ▷ — Generator Update —
    opt_G.zero_grad()
    if $\mathscr{C}$ has learnable parameters **then**
      $\text{opt}_{\mathscr{C}}$.zero_grad()
    end if
    $L_G^{\text{adv}} \leftarrow \ell_{\text{BCE}}(D(x_G), y_{\text{real}})$
    $L_{\text{aux}} \leftarrow \mathscr{A}(x_G)$ ▷ Optional, if $\mathscr{A}$ is defined
    $L_G \leftarrow \alpha L_G^{\text{adv}} + \lambda L_{\text{constraint}} + \beta L_{\text{aux}}$
    backprop L_G, opt_G.step()
    if $\mathscr{C}$ has learnable parameters **then**
      $\text{opt}_{\mathscr{C}}$.step()
    end if
    Log metrics: L_G, L_G^{adv}, L_D, S_{logic}, λ, $L_{\text{constraint}}$, discriminator accuracy
  end for
end for
return G^*, D^*

[1] **Loss Definitions.** Let ℓ_{BCE} denote the binary cross-entropy loss. Discriminator losses: $L_D^{\text{real}} = \mathbb{E}_{x_R}[\ell_{\text{BCE}}(D(x_R), y_{\text{real}})]$, $L_D^{\text{fake}} = \mathbb{E}_z[\ell_{\text{BCE}}(D(G(z)), y_{\text{fake}})]$, with total $L_D = L_D^{\text{real}} + L_D^{\text{fake}}$.

Generator losses include: $L_G^{\text{adv}} = \mathbb{E}_z[\ell_{\text{BCE}}(D(G(z)), y_{\text{real}})]$, $L_{\text{constraint}} = 1 - S_{\text{logic}}$ (or adaptive weighted sum), and optional L_{aux} for classification, template matching, or other tasks.

The overall generator loss is $L_G = \alpha L_G^{\text{adv}} + \lambda(e) L_{\text{constraint}} + \beta L_{\text{aux}}$, where α, β, and $\lambda(e)$ modulate the loss components.

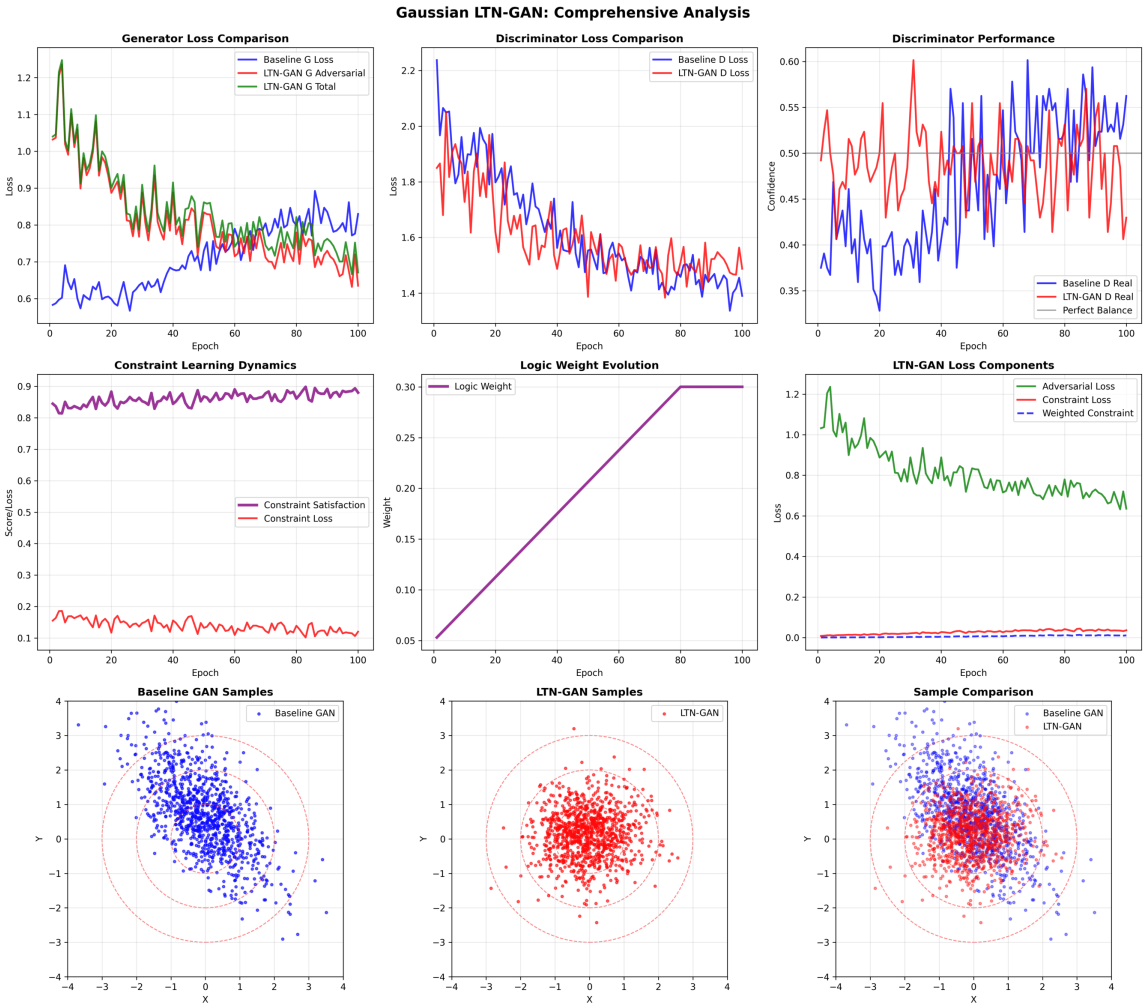
Logic Weight Schedule. $\lambda(e)$ increases linearly over the first K epochs:

$$\lambda(e) = \begin{cases} \lambda_{\text{start}} + \dfrac{e-1}{K-1}(\lambda_{\text{end}} - \lambda_{\text{start}}), & e \le K \\ \lambda_{\text{end}}, & \text{otherwise} \end{cases}$$

[2] **Dataset Notes:** Gaussian and Ring use linear λ ramps. Grid logs both $L_{\text{constraint}}$ and $\lambda L_{\text{constraint}}$. Ring employs hierarchical and adaptive constraints. MNIST includes an auxiliary loss L_{aux} for classification and shape templates. No early stopping is used. All experiments compute L_D as a plain sum (no $\frac{1}{2}$ scaling).

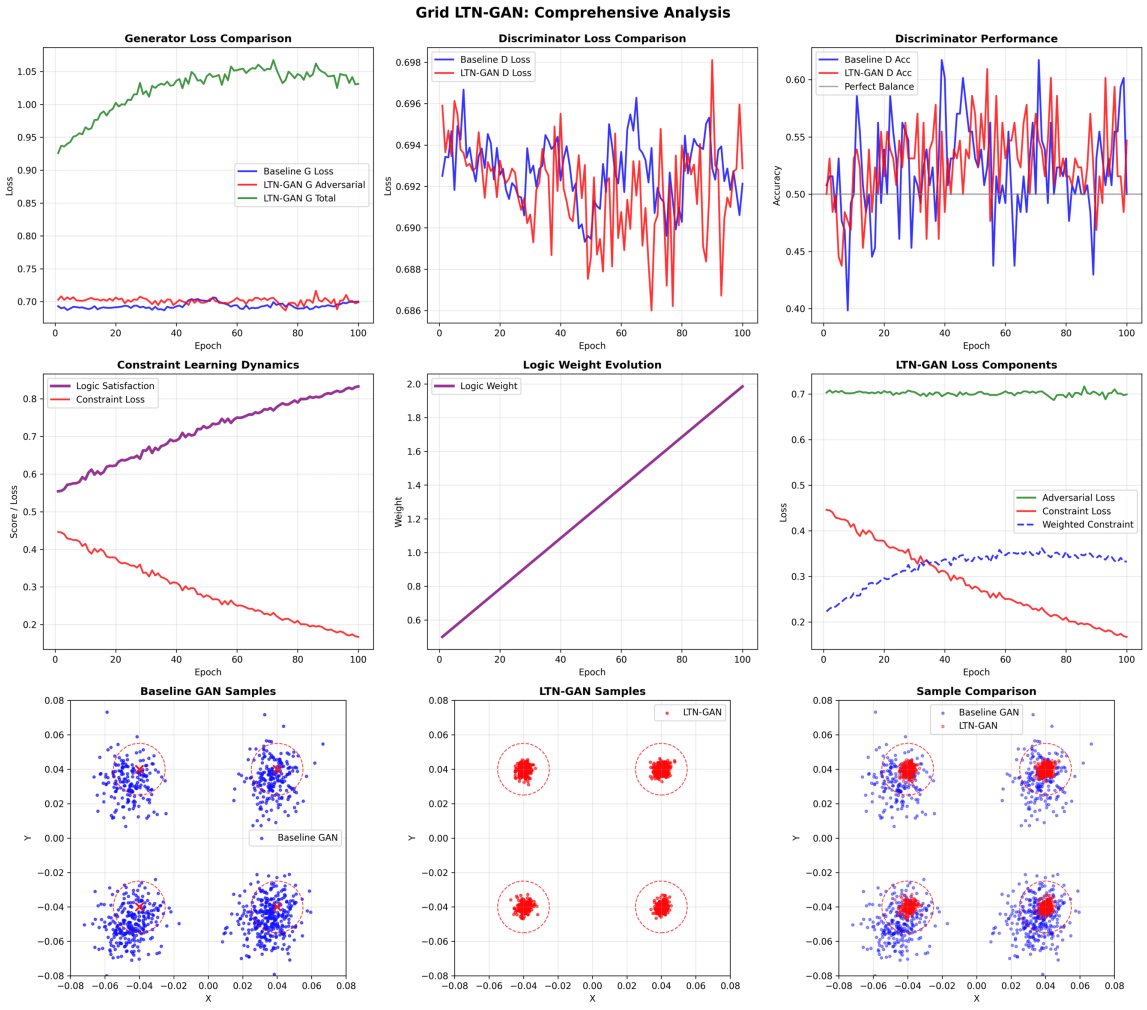
B Experimental Results and Visualizations

B.1 Gaussian Dataset



For the Gaussian experiment, both predicates—$\texttt{InRange}(x)$ and $\texttt{GaussianShape}(x)$—are implemented as fixed mathematical functions rather than learned neural networks, enabling symbolic supervision without additional learning overhead. The generator is a fully connected MLP with LeakyReLU activations and no output activation, allowing it to generate samples across the full $\mathbb{R}^2$ plane. To stabilize training, label smoothing is applied (real samples labeled as 0.9, fake as 0.1), and the logic loss weight $\lambda(e)$ increases linearly from 0.05 to 0.3 over the first 80 epochs. Formula satisfaction is computed as the average truth degree of predicate conjunctions across the batch. Since all constraints are geometric and explicitly defined, the Gaussian task serves as a controlled testbed to isolate the effects of symbolic logic on GAN training without involving predicate learning.

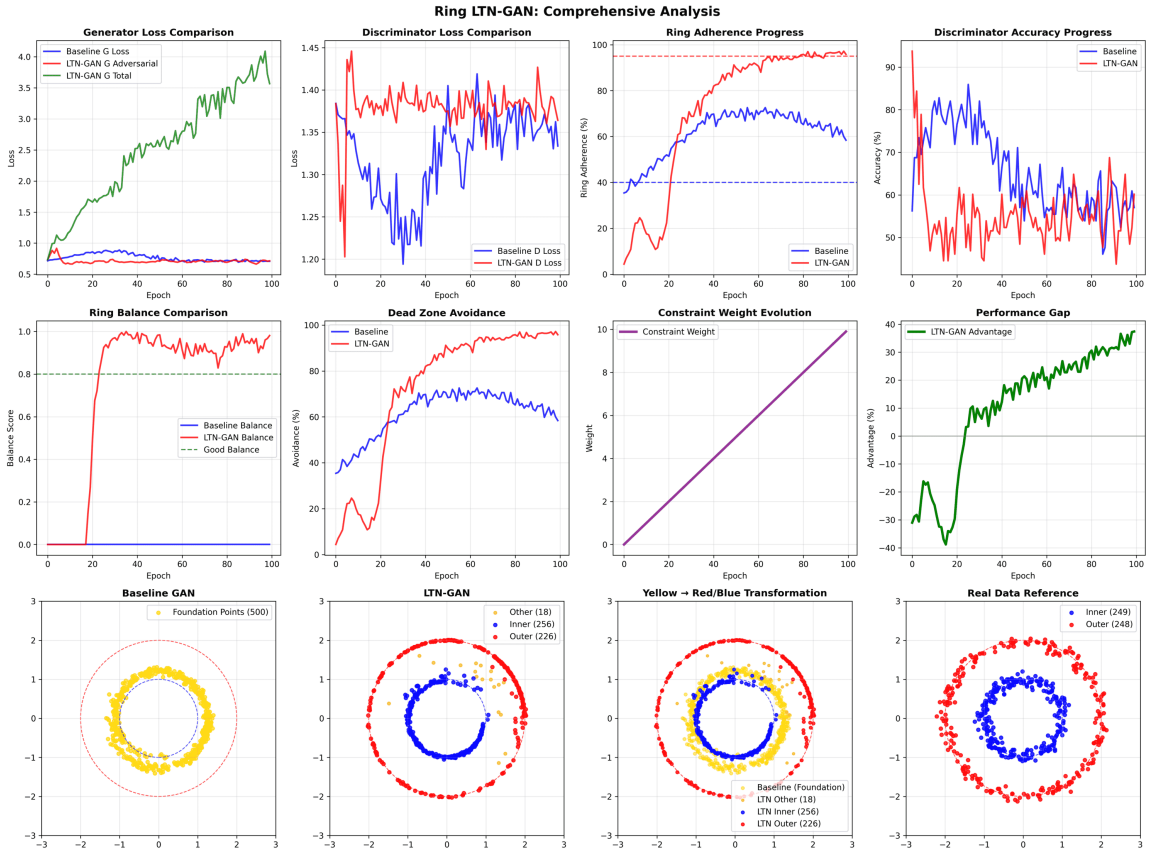
B.2 Grid Dataset



In the Grid experiment, the generator is a fully connected MLP with LeakyReLU activations and no output activation, enabling unconstrained 2D outputs. A categorical selector network assigns each latent vector to one of four grid cells, and a perturbation is applied around that cell's center to yield the final coordinate. This inductive bias guides initial structure formation.

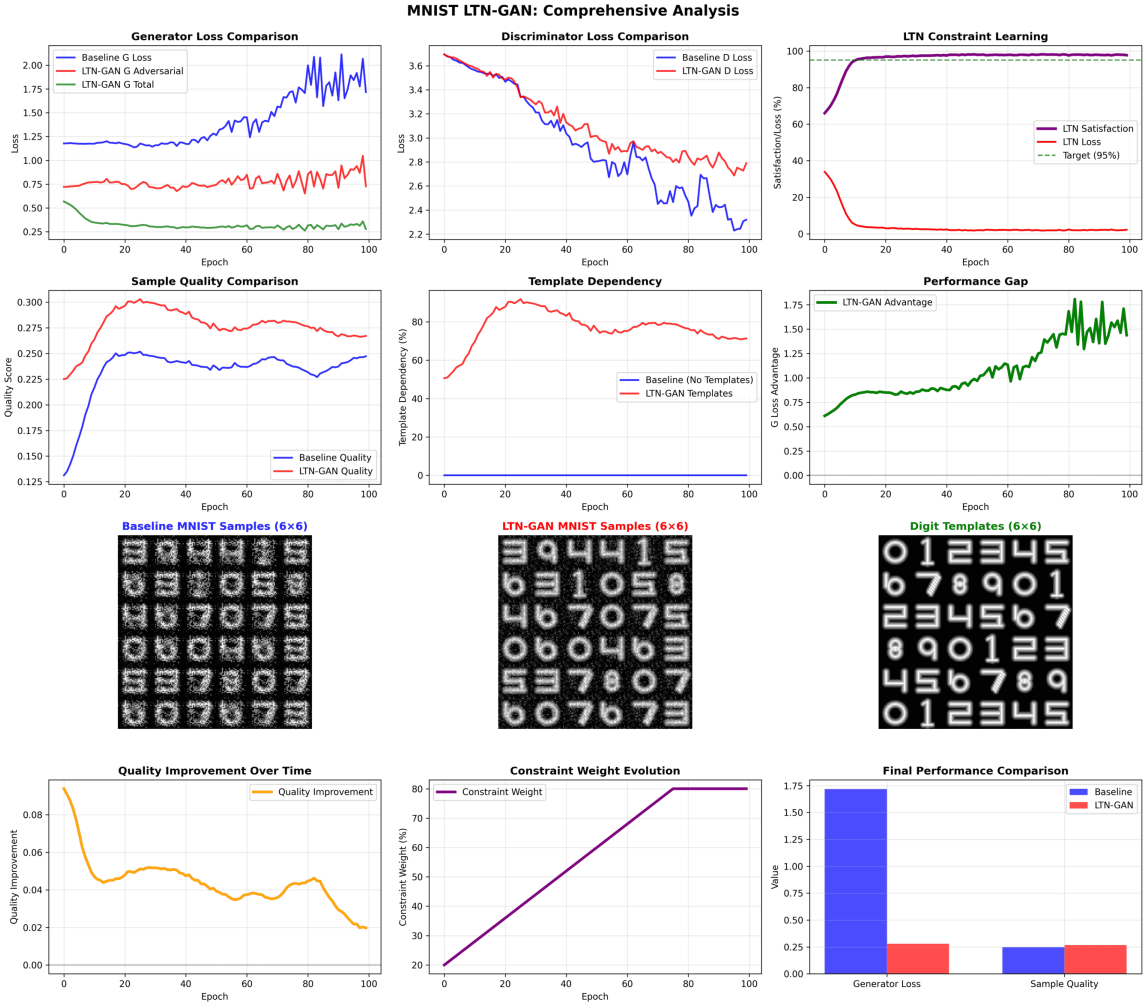
LTN predicates include a shared $\texttt{OnGrid}(x)$ network, evaluating proximity to any grid center, and four $\texttt{InCell}_i(x)$ networks, detecting cell-specific membership. These are trained jointly using distance-based features. Logical supervision is applied via a universal constraint over $\texttt{OnGrid}(x)$ and existential constraints over each $\texttt{InCell}_i(x)$, ensuring both alignment and coverage. The logic loss weight $\lambda(e)$ increases linearly from 0.5 to 2.0 over 100 epochs. Real samples are synthetic perturbations of grid centers (noise $\sigma = 0.008$), and label smoothing (0.9/0.1) is used to stabilize training.

B.3 Rings Dataset



In the Ring experiment, the data is sampled in polar coordinates and converted to Cartesian space, allowing for controlled radial distributions. Hardcoded geometric priors guide the generator to favor annular manifolds corresponding to inner and outer rings, while the dead zone in between is left unsampled. The generator is a multilayer perceptron with LeakyReLU activations and Tanh output to constrain the output space. Unlike in the Gaussian setting, most predicates—including `InnerRing`, `OuterRing`, `DeadZone`, and `NearCenter`—are implemented as differentiable neural classifiers trained jointly to provide flexible fuzzy memberships. A dynamic rule reweighting strategy updates clause weights based on per-batch satisfaction rates, allowing the model to adaptively prioritize under-satisfied constraints. The logic loss schedule begins at $\lambda = 0.01$ and gradually increases to $\lambda = 0.2$, with predicate tolerances (e.g., ring radius bands) tightened after warmup epochs to progressively refine precision.

B.4 MNIST Dataset



In the MNIST experiment, the generator is a convolutional neural network with transposed convolutions and batch normalization, trained to synthesize 28×28 grayscale digit images. Class predicates $\texttt{IsDigit}_k(x)$ for $k \in \{0,\ldots,9\}$ are implemented as frozen MLPs trained on real MNIST digits, allowing symbolic supervision over digit identity. In addition, four structural quality predicates—$\texttt{ValidPixels}(x)$, $\texttt{IsConnected}(x)$, $\texttt{IsComplete}(x)$, and $\texttt{HasProperIntensity}(x)$—are defined using simple image processing heuristics and shallow neural nets. A handcrafted dataset of binary digit templates is used to blend geometric priors into the early training stages, guiding initial sample formation before symbolic constraints take effect. Logic loss weighting $\lambda(e)$ increases from 0.1 to 0.5 over the first 40 epochs. All predicates are evaluated on generated batches and their logic satisfaction scores are combined using the weighted fuzzy logic aggregation. This setup enables LTN-GAN to generate structurally valid and class-consistent images while preserving diversity across digits. MNIST predicate networks are shallow CNNs trained end-to-end to capture digit properties such as centering and shape. Early training includes a blending of template priors, which are gradually phased out to shift supervision toward symbolic rules.

B.5 Domain-specific Metrics

Dataset	Metric	Description / Computation
Gaussian	Gaussian Adherence	Composite score based on stringent tests: normality (Anderson-Darling), bivariate shape (Mahalanobis distance), moments (mean ≈ 0, std ≈ 1), radial distribution (Rayleigh fit), independence (Pearson corr), and quartiles. Penalized if logic satisfaction is low.
	Statistical Quality	Inverse of total deviation from target mean and std: $1/(1 + \|\mu\| + \|\sigma - 1\|)$
	Combined Quality	Arithmetic mean of adherence and statistical quality.
	Mean Error	L2 norm of difference between sample mean and 0.
	Std Error	L2 norm of difference between sample std and 1.
	Logic Satisfaction	Average truth degree of LTN constraints (range and Gaussianity).
Grid	Grid Cluster	Fraction of samples that belong to valid grid cluster centers (e.g., KMeans+thresholding on center match).
	Coverage	Ratio of grid positions covered at least once.
	Quality	Precision-like score evaluating how many points fall within valid tolerance of expected clusters.
	Overall Performance	Weighted average of coverage and quality.
	In Targets	Number of points within valid grid zones (rounded to integers or spatial masks).
	Logic Satisfaction	Mean satisfaction across structural LTN constraints.
Ring	Ring Adherence	Fraction of samples lying near ring regions, measured via radial distance scoring and circular symmetry.
	Inner / Outer Count	Number of samples classified inside or outside ring boundaries.
	Balance Score	Normalized measure of symmetry between inner and outer zones (e.g., $1 - \|n_{in} - n_{out}\|/(n_{in} + n_{out})$).
	Dead Zone Avoidance	Fraction of samples not falling into undesired central dead region.
	Logic Satisfaction	Mean constraint satisfaction from LTN logic over spatial rings.
MNIST	Quality	Standard GAN image fidelity metrics (can include Inception-like proxy or visual diversity).
	Coverage	Fraction of target digits generated (using classifier or template matching).
	Digit Recognition	Accuracy of trained classifier over generated samples.
	Template Dependence	Score measuring how many digits matched predefined templates.
	Logic Satisfaction	Satisfaction over digit identity and position constraints defined in LTN logic.

Table 6: Overview of evaluation metrics used across dataset-specific ablation studies. Each metric is designed to align with both generative quality and logical structure fidelity.

Interpretable Hybrid Machine Learning Models Using FOLD-R++ and Answer Set Programming

Sanne Wielinga
Open Universiteit, the Netherlands
sannwielinga@gmail.com

Jesse Heyninck
Open Universiteit
Heerlen, Limburg, the Netherlands
University of Cape Town and CAIR
Cape Town, South Africa
jesse.heyninck@ou.nl

Machine learning (ML) techniques play a pivotal role in high-stakes domains such as healthcare, where accurate predictions can greatly enhance decision-making. However, most high-performing methods such as neural networks and ensemble methods are often opaque, limiting trust and broader adoption. In parallel, symbolic methods like Answer Set Programming (ASP) offer the possibility of interpretable logical rules but do not always match the predictive power of ML models. This paper proposes a hybrid approach that integrates ASP-derived rules from the FOLD-R++ algorithm with black-box ML classifiers to selectively correct uncertain predictions and provide human-readable explanations. Experiments on five medical datasets reveal statistically significant performance gains in accuracy and F1 score. This study underscores the potential of combining symbolic reasoning with conventional ML to achieve high interpretability without sacrificing accuracy.

1 Introduction

Machine learning has become an essential tool for predictive analytics and decision-making in various domains, including healthcare [TJMG19]. Advanced ML models, particularly black-box models such as neural networks and ensemble methods, have shown impressive predictive performance. However, many of these high-performing models offer little insight into *why* a specific prediction is made. This lack of transparency slows down the adoption of models in applications where understanding the reasoning behind a prediction is as important as the prediction itself [ADRDS+20]. In fields like medicine, decisions informed by ML models can have serious implications on patient outcomes. The ability to explain and justify predictions is therefore important for trust, accountability, and informed decision-making [DVK17].

Meanwhile, symbolic methods such as Answer Set Programming (in short, ASP), allow to represent knowledge in transparent, human-understandable logical rules [EGL16]. The FOLD-R++ algorithm uses ASP to learn default rules [WG22]. These rules can complement black-box models by offering explanations for their predictions and therefore improving interpretability.

However, integrating ASP-derived rules with black-box ML models without changing their internal mechanisms remains challenging. Hybrid neuro-symbolic methods have emerged as a promising solution, though most of them emphasize neural architectures and require specialized integration [GGL+19, MDK+18, YIL23]. In contrast, relatively few works integrate symbolic rules with other types of black-box models, such as support vector machines or ensemble methods.

This paper proposes a hybrid approach that addresses this gap by integrating FOLD-R++, an algorithm that learns ASP rules with exceptions [WG22], with various black-box ML models. By using rules

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 114–127, doi:10.4204/EPTCS.439.9

© S. Wielinga and J. Heyninck
This work is licensed under the
Creative Commons Attribution License.

that capture domain knowledge, the hybrid system can (1) correct ML model errors when the model is uncertain and (2) provide human-readable explanations.

The following research questions guide this study:

RQ1: Does integrating interpretable ASP rules derived from FOLD-R++ improve the predictive performance of black-box ML models across various medical datasets?

RQ2: How does the hybrid model of ML and ASP improve the interpretability of predictions?

The remainder of this paper is organized as follows: Section 2 provides an overview of interpretability research and other related approaches. Section 3 covers the background on ASP syntax and semantics, the FOLD-R++ algorithm, and a brief discussion of traditional ML methods. Section 4 outlines the methodology, including data preparation, model training, and hybrid model implementation. Section 5 presents the empirical findings. Section 6 concludes with limitations and paths for future research.

2 Related Work

2.1 Interpretability and Related Approaches

The need for interpretable ML has surged amid concerns for trust, transparency, and accountability [GMR+18]. Traditional interpretable models, such as decision trees and linear models, provide clear decision paths but may miss complex interactions in large datasets. In contrast, advanced ML techniques, such as deep neural networks and ensemble methods, achieve superior performance but function as "black-box" models with opaque decision-making processes.. This opacity is problematic in areas such as healthcare, where interpretability is essential for ethical considerations, regulatory compliance, and establishing trust among clinicians and patients [TJMG19]. Clinicians must understand how a model arrived at a specific diagnosis to trust and effectively use it in decision-making [TJMG19]. Lack of transparency can lead to mistrust or rejection of ML systems [ADRDS+20]. Moreover, regulatory bodies increasingly require explanations for automated decisions, especially when they impact patient care [PLV24].

Researchers have proposed various strategies to address these challenges. These include developing inherently interpretable models such as decision trees and rule-based systems [Rud19], and using post-hoc techniques like SHAP [Lun17] to explain predictions from black-box models. These methods attempt to approximate the behavior of complex models locally with simpler models. However, a trade-off exists between interpretability and performance: simple models may miss complex patterns, while complex models remain opaque. Our proposed hybrid method aims to navigate this trade-off by integrating interpretable rules with high-performing black box models.

2.2 Hybrid Models

A key line of related work focuses on combining symbolic reasoning and machine learning, which focuses on integrating symbolic reasoning with neural networks, creating neuro-symbolic systems [GGL+19]. For instance, DeepProbLog integrates probabilistic logic programming with deep learning [MDK+18]. NeurASP is a framework that combines neural networks with ASP [YIL23].

These approaches often require modifications to the learning algorithms or network architectures, which can be complex and computationally intensive. Moreover, they primarily focus on neural networks and do not address integration with other types of black-box ML models, such as random forests or support vector machines. Thus, a gap remains on the *model-agnostic* integration of symbolic reasoning.

This study addresses this gap by integrating interpretable ASP rules from FOLD-R++ with a variety of black-box ML classifiers. The hybrid approach does not require changing the ML models, thus preserving their performance. The ASP component adds a layer of interpretability by offering human-understandable explanations for the predictions.

3 Background

3.1 Symbolic Logic and Answer Set Programming

Symbolic logic offers a framework for interpretable models by encoding knowledge in structured, human-readable formats. Answer Set Programming combines logic programming with non-monotonic reasoning to represent complex relationships and constraints [EGL16]. ASP programs consist of rules, which in turn are composed of literals (positive or negated atomic formulas) and constraints that define which combinations of literals cannot all be true simultaneously (integrity constraints). A key aspect of ASP is its use of stable model or answer set semantics to determine which sets of ground literals are considered stable solutions to a program. The non-monotonic nature of ASP makes it ideal for representing default rules and exceptions. An answer set program II consists of rules of the form:

a_0 :- a_1, ..., a_m, not a_{m+1}, ..., not a_n.

where each a_i is an atom, and not represents negation as failure. The head of the rule (a_0) is derived if all positive literals in the body (a_1, ...,a_m) are true and all negated literals (not a_{m+1},..., not a_n) are not proven true. An answer set is a minimal set of literals satisfying all program rules without cyclic support. Multiple answer sets represent different possible solutions to the encoded problem. Solvers like clingo [GKKS19] can automatically calculate these answer sets.

ASP is particularly valuable for tasks requiring interpretability. Applications of ASP can be found in many areas, including planning, scheduling, and bio-informatics [EGL16]. In healthcare specifically, ASP has been used to model clinical guidelines [STD17] and solve scheduling problems [EGL16].

3.2 FOLD-R++ Algorithm

The FOLD-R++ algorithm [WG22] extends the First-Order Logical Decision tree (FOLD) algorithm to learn default rules with exceptions from relational data, producing output that is both human-readable and compatible with ASP reasoning. FOLD-R++ generates default rules for general patterns while capturing exceptions for special cases or anomalies. It operates by recursively partitioning the data to construct a decision tree (similar to algorithms like ID3 or C4.5), but transforms the decision tree into an ASP.

The algorithm starts with the complete dataset and evaluates all possible literals (attribute-value pairs or relational literals) that could split the data. At each node, it selects the literal that best separates positive from negative examples according to a heuristic, incorporating this literal into the rule condition. When a rule cannot perfectly classify the data, the algorithm identifies exceptions to the rule and induces them as sub-rules through the same recursive process. Mechanisms for pruning unnecessary rules or exceptions are included to prevent overfitting. The final ruleset is translated into an ASP program.

FOLD-R++-induced rules have the following form:

```
label(X, Class) :- Conditions(X), not Exceptions(X).
```

where `Conditions(X)` represent the conditions under which an instance `X` belongs to a particular class `Class`, and `Exceptions(X)` are exceptions to this rule.

We chose FOLD-R++ for its proven effectiveness and compatibility with the experimental setup. Although newer algorithms in the same family, like FOLD-SE and FOLD-RM, offer advanced features, they were not selected due to accessibility constraints and alignment with the research focus.

3.3 Traditional ML Methods

In addition to FOLD-R++, this paper considers four widely used machine learning classifiers:

- **Random Forest (RF):** An ensemble method building multiple decision trees using bootstrap aggregation and random feature selection, reducing variance and improving generalization.
- **Support Vector Machine (SVM):** A model that finds the optimal hyperplane that best separates classes in a high-dimensional space.
- **K-Nearest Neighbors (KNN):** A non-parametric method that classifies instances based on the majority class among the k-nearest neighbors in the feature space.
- **Neural Network (MLPClassifier):** A multi-layer perceptron that learns complex non-linear relationships through backpropagation.

These models represent a spectrum of approaches in modern machine learning, from relatively transparent (KNN) to highly opaque (neural networks). Each has strengths and weaknesses in terms of predictive performance, generalization ability, and interpretability. Of course, this is just a selection of ML-methods, and our framework allows for the integration of other ML-classifiers as well.

4 Methods

This section outlines the methodology used to develop and evaluate the hybrid models that combine black-box ML models with interpretable rules generated by FOLD-R++ using ASP. The general pipeline consists of data preparation, ML model training, rule induction via FOLD-R++, and final integration of predictions using a confidence threshold. Figure 1 provides an overview of this pipeline. Relevant data and code can be found online: `https://github.com/sannewielinga/foldrpp-asp-hybrid`.

4.1 Data Preparation

Five medical datasets were selected from the UCI Machine Learning Repository [KLN25]:

- **Heart Disease:** Contains 303 instances with 14 commonly used attributes. It is used to predict the presence of heart disease in patients based on medical measurements.
- **Autism Screening Adult:** Includes 704 instances with 21 attributes, used to predict the likelihood of autism spectrum disorder based on screening test scores and demographic information.
- **Breast Cancer Wisconsin:** Consists of 569 instances with 30 numerical features computed from digitized images. The goal is to classify tumors as malignant or benign.
- **Ecoli:** Contains 336 instances with 8 attributes, used to classify protein localization sites.
- **Chronic Kidney Disease:** Contains 400 instances with 24 attributes, used to predict the presence of chronic kidney disease in patients based on clinical and laboratory findings.

Each dataset underwent preprocessing to handle missing values, encode categorical variables, and scale numerical features where necessary. Stratified sampling was used to split the datasets into training and testing sets to maintain class distribution. A different random seed was used for each experiment.

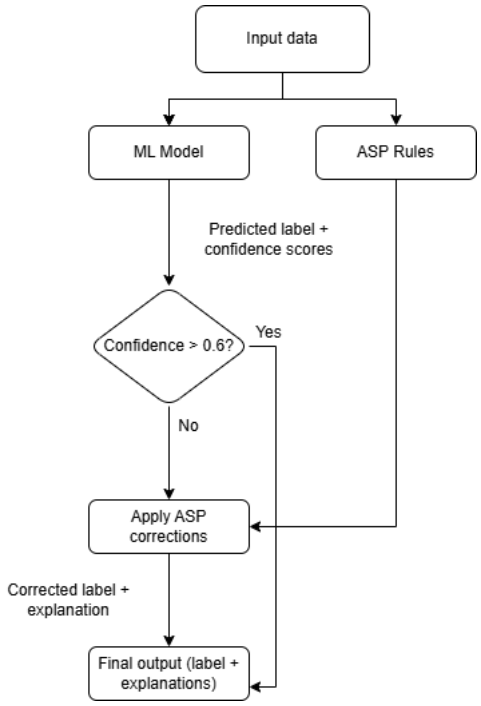


Figure 1: Conceptual diagram of the hybrid model architecture. Input data is processed by both the black-box ML model and the ASP rule-based component. The confidence score determines whether the ML prediction is accepted directly or corrected by the ASP rules.

4.2 FOLD-R++

FOLD-R++ [WG22] was used to induce ASP rules from the training data. The original FOLD-R++ code was used and slightly refactored to fit the experimental setup. This included adding new functions to transform data into ASP-compatible formats and interface with Clingo [GKKS19]. Additionally, a wrapper was created to handle the training of FOLD-R++, conversion of induced rules to ASP syntax, and prediction using Clingo. The training data was transformed into a format compatible with FOLD-R++, where each instance was represented as a set of logical facts with attributes and their values forming predicates. For example, an attribute-value pair such as `age = 45` was converted into a predicate `age(X,45)`.

The induced rules were converted into ASP syntax compatible with the Clingo solver. Numeric values were scaled by a factor 10 to handle the limitations of Clingo with floating-point numbers. The scaling was applied consistently to keep relationships between variables valid. Clingo was then used to apply the logical rules to the test data, reasoning over the facts representing each instance and the induced rules to infer the class labels.

4.3 ML Models

Four black-box ML models were trained for each dataset: Random Forest, SVM, KNN, and MLPClassifier (see Section 3.3). Default hyperparameters were used for all models unless adjustments were necessary for convergence or performance. Random seeds were set for model initialization and data splitting to

ensure reproducibility across experiments. For models that provide probability estimates (e.g., RF, SVM with `probability=True`, MLP), the `predict_proba` method was used to obtain confidence scores for each class. For KNN, probabilities were derived from the proportion of neighbors belonging to each class.

4.4 Hybrid Model Implementation

The hybrid model integrates predictions from the ML models with the interpretable rules generated by FOLD-R++ using ASP. First, predictions and confidence scores from the ML model on the test set were obtained. Clingo was then used to apply the induced rules to the test instances and generate predictions.

A fixed 0.6 confidence threshold, empirically selected from preliminary evaluations, was used to determine when to rely on the prediction of the ML model versus the ASP rules. If the confidence score of the ML model for a prediction was above 0.6, the ML prediction was used; otherwise, the prediction of the ASP rule was used. This approach ensures that the ASP rules intervene specifically when the ML model is uncertain. During development, both dynamic and fixed confidence thresholds were experimented with, but the dynamic threshold led to overfitting on the training data. In practice, this threshold can be tuned based on domain needs.

For each prediction corrected by the ASP rules, explanations were generated from the induced rules. These explanations highlight which logical conditions were met and which exceptions were not triggered. Figure 1 provides a schematic representation of the hybrid model architecture.

4.5 Experimental Setup

Ten experiments were conducted for each combination of dataset and model. Each experiment used stratified sampling for an 80-20 train-test split with varying random seeds to alter splits and model initialization. These represented repeated random subsamples rather than traditional k-fold cross-validation, testing result robustness through split variability. For each experiment, accuracy, precision, recall, and F1 score (the harmonic mean of precision and recall) were computed and averaged across the ten experiments (with standard deviations calculated).

Paired t-tests were conducted to determine whether the improvements observed with the hybrid model were statistically significant compared to the standalone ML models. A p-value threshold of 0.05 was used to determine statistical significance. If the ML model and hybrid model had the same accuracy in all experiments, the differences were zero, and the paired t-test could not be performed.

The importance of each ASP rule was determined based on how frequently it corrected errors made by the ML models. Due to variability in the induced ASP programs across different experiments - likely caused by randomness in data splitting - only the ASP programs and important rules from model-dataset combinations that showed statistically significant improvements were saved. Explanations were generated for instances where the hybrid model corrected the prediction of the ML model, using proof trees from the FOLD-R++ algorithm. For example, in one instance from the Heart Disease dataset (`patient5`), the SVM model predicted a positive label (indicating heart disease), which the hybrid model corrected this to a negative prediction. The FOLD-R++ explanation is summarized as follows:

• **Extracted Features:** `{'major_vessels': '2', 'chest_pain': '4', 'age': 60.0, 'slope': '2', 'blood_pressure': 150.0, 'oldpeak': 2.6, 'thal': '7', 'exercise_induced_angina': '0', 'maximum_heart_rate_achieved': 157.0}`
• **Key Proof Tree Points:**

1. "label(X,'absent') does not hold because the value of `major_vessels` is '2' which should equal '0' does not hold."
2. "label(X,'absent') does not hold because the value of `chest_pain` is '4' which should not equal '4' does not hold and the value of `age` is 60.0 which should be greater than 66.0 does not hold."

The implementation was done using Python, with libraries such as scikit-learn for ML models, SciPy for statistical tests, and clingo for ASP solving. Pandas and NumPy were used for data manipulation and numerical computations. The FOLD-R++ library was used to induce logical rules from data. The project was modularized into separate components for data loading, model training, hybrid integration, and evaluation to improve readability.

5 Results

This section presents the experimental results of evaluating the hybrid models that integrate black-box ML classifiers with interpretable rules derived from FOLD-R++ using ASP. The performance of the ML models and the hybrid model is compared across multiple medical datasets. Additionally, the role of ASP in improving interpretability and correcting errors is illustrated through case studies.

Table 1 summarizes the average accuracy and F1 score of the ML models, FOLD-R++ used in isolation and the hybrid models across all datasets and experiments.

The results indicate that the hybrid models demonstrated an improvement in accuracy and F1 score over the standalone ML models across several datasets. Paired t-tests, presented in Table 1, were conducted to determine whether these observed improvements were statistically significant. If the ML model and hybrid model have the same accuracy in all experiments, the differences were zero, and the paired t-test could not be performed (denoted as "N/A"). The SVM classifier, in particular, showed significant improvements. When compared to the pure FOLD-R++ approach, which is applied uniformly across each dataset, the hybrid models can significantly enhance performance for classifiers with lower baseline scores. Conversely, when a baseline model already achieved near-perfect scores, the hybrid approach typically maintained similar performance.

Overall, these findings suggest that the hybrid approach is particularly beneficial for cases where the ML baseline is suboptimal, although the optimal method remains dependent on the dataset.

5.1 Results by Dataset

In the following subsections, examples are presented from four representative datasets. For each, we highlight improved predictions (where the ASP rules successfully corrected a misclassification), no-change predictions, and worsened predictions (where the ASP rules overrode a correct classification). For example, a misclassified instance may have triggered a learned ASP rule that encodes clinically relevant conditions, thus overriding a low-confidence ML guess. Conversely, when the conditions or exceptions of a rule were misaligned with the true label, the override led to a worse outcome. Finally, in some high-confidence cases, the model's prediction was already correct and thus remained unchanged. All examples come from the explanation files generated during the experiments. Detailed analyses for the remaining dataset can be found in Appendix A.

5.1.1 Heart Disease

The hybrid model improved the accuracy of the SVM classifier from 63.5% to 71.3%. In particular, the first rule in Listing 1 shows the ASP rule for labeling a case as *absent* (i.e., no heart disease) when

Model	ML Acc	Hybrid Acc	ML F1	Hybrid F1	t-statistic	p-value	Significant
Heart Disease							
FOLD-R++	74.26	-	76.39	-	-	-	-
KNN	64.26	64.26	68.97	68.97	N/A	N/A	N/A
MLP	80.93	80.19	81.22	80.65	-0.53	0.61	No
RF	79.44	79.44	81.24	81.39	0.00	1.00	No
SVM	63.52	**71.30**	70.96	**75.14**	3.55	0.01	Yes
Autism Screening							
FOLD-R++	93.40	-	95.49	-	-	-	-
KNN	87.38	87.38	91.52	91.52	N/A	N/A	N/A
MLP	97.52	97.66	98.30	98.29	1.00	0.34	No
RF	97.38	96.10	98.20	97.33	-3.04	0.01	Yes
SVM	72.62	**94.04**	84.13	**96.01**	32.12	1.35×10^{-10}	Yes
Breast Cancer Wisconsin							
FOLD-R++	94.43	-	92.04	-	-	-	-
KNN	94.36	94.36	91.62	91.62	N/A	N/A	N/A
MLP	93.57	**94.43**	91.04	**92.04**	3.09	0.01	Yes
RF	96.36	95.50	94.78	93.63	-2.54	0.03	Yes
SVM	95.21	95.36	93.11	93.28	0.80	0.44	No
Ecoli							
FOLD-R++	95.15	-	94.58	-	-	-	-
KNN	65.74	65.74	57.46	57.46	N/A	N/A	N/A
MLP	92.50	94.71	91.14	93.93	2.75	0.02	Yes
RF	96.32	96.47	95.77	95.96	0.36	0.73	No
SVM	57.06	**87.50**	23.37	**82.99**	7.30	4.58×10^{-5}	Yes
Chronic Kidney Disease							
FOLD-R++	62.00	-	76.01	-	-	-	-
KNN	89.13	89.13	90.36	90.36	N/A	N/A	N/A
MLP	95.25	95.50	96.05	96.28	0.69	0.51	No
RF	100.00	99.88	100.00	99.89	-1.00	0.34	No
SVM	91.38	**93.13**	92.60	**94.27**	3.74	0.01	Yes

Table 1: Combined results (mean values only) for Accuracy (%) and F1 Score (%) across five medical datasets. The constant FOLD-R++ performance for each dataset is indicated in the header. Bold entries highlight the model with the best improvement within each dataset. Statistical significance ($p < 0.05$) from paired t-tests comparing ML and Hybrid performance is shown on the right (N/A indicates no difference).

```
label(X, absent) :- thal(X, 3), maximum_heart_rate_achieved(X, V_max_hr_1),
V_max_hr_1 > 71.0, not ab2(X, True), not ab3(X, True), not ab4(X, True), not ab5(X,
    True), not ab6(X, True), not ab7(X, True), not ab8(X, True).
ab2(X, True) :- chest_pain(X, 4), major_vessels(X, V_major_vessels_1),
    V_major_vessels_1 != 0, not ab1(X, True).
```

Listing 1: ASP Rules for Heart Disease.

```
label(X, NO) :- a5(X, V_a5_0), V_a5_0 != 1, not ab1(X, True), not ab2(X, True).

ab1(X, True) :- a9(X, 1), a3(X, 1), a1(X, 1), a6(X, 1).
```

Listing 2: ASP Rule for Autism Screening.

`thal` equals 3 and the maximum heart rate exceeds 71, provided no exceptions apply. The second rule in Listing 1 defines one such exception (`ab2`), which triggers if the patient exhibits chest pain type 4 and has a nonzero number of major vessels affected.

An illustrative example is `patient34`, who was originally predicted "absent" by the SVM model. After applying the rule in Listing 1, the hybrid model considered potential exceptions and corrected the prediction to "present". The patient had `thal` equal to 3 and a maximum heart rate achieved of 150 (greater than 71), but exceptions did not hold due to the absence of chest pain type 4 and no major vessels affected. This alignment with clinical indicators led to a more accurate diagnosis.

There are cases where the ASP rules did not alter an ML prediction when both the model and the learned rules align on the same label. For example, in one instance the SVM model predicted "present" with high confidence, and the rules confirmed that label without any exceptions triggered. However, the ASP rules can sometimes degrade performance if they override an already correct ML prediction. This occurs when a learned rule is too general or when exceptions fail to capture the subtle conditions that the ML model detected. In one such instance, the SVM model correctly predicted "present", but the hybrid approach changed it to "absent" because a relevant exception in the rule set was not satisfied.

5.1.2 Autism Screening

The SVM classifier showed significant improvement when combined with ASP rules, with accuracy increasing from 72.6% to 94.0%. However, performance slightly decreased when hybridized with ASP, likely because of the original high accuracy of the model and the rules occasionally overriding correct predictions. Listing 2 presents the rule for labeling an individual as `NO` (not autistic) if `a5` is not equal to 1 and exceptions `ab1`, `ab2` do not apply. Listing 2 also outlines the exception rule `ab1`, which captures strong autistic traits indicated by affirmative answers to certain questions.

For instance, in the case of `patient1`, the ML model predicted "autistic". However, the rule in Listing 2 applied, and neither `ab1` nor `ab2` was valid. Consequently, the hybrid model overruled the ML model and correctly classified `patient1` as "non-autistic".

Similarly, some instances that the ML model correctly classified as "autistic" were left unchanged by the ASP rules. However, there are also examples in which a correct ML prediction was overridden incorrectly by a default rule or a missing exception. This can happen if the rule induction process relies on partial indicators that conflict with more nuanced patterns captured by the classifier. As a result, the hybrid system incorrectly switched the label to "non-autistic".

```
label(X, malignant) :- cell_size_uniformity(X, V_cell_size_uniformity_0),
    V_cell_size_uniformity_0 != 1, not ab1(X, True), not ab2(X, True), not ab3(X, True
    ).
ab1(X, True) :- bare_nuclei(X, 1), cell_size_uniformity(X, V_cell_size_uniformity_1),
    V_cell_size_uniformity_1 != 10, marginal_adhesion(X, V_marginal_adhesion_2),
    V_marginal_adhesion_2 != 10, clump_thickness(X, V_clump_thickness_3),
    V_clump_thickness_3 != 7, cell_size_uniformity(X, V_cell_size_uniformity_4),
    V_cell_size_uniformity_4 != 6, bland_chromatin(X, V_bland_chromatin_5),
    V_bland_chromatin_5 != 5.
```

Listing 3: ASP Rule for Breast Cancer.

```
label(X, cp) :- sn(X, V_sn_0), V_sn_0 != FECR, alm1(X, V_alm1_1), V_alm1_1 <= 0.38,
    not ab1(X, True).

ab1(X, True) :- sn(X, V_sn_0), V_sn_0 != PTKB, gvh(X, V_gvh_1), V_gvh_1 > 0.55, mcg(X,
     V_mcg_2), V_mcg_2 > 0.41.
```

Listing 4: ASP Rule for Ecoli.

5.1.3 Breast Cancer Wisconsin

The performance of the MLP classifier improved with the hybrid model, increasing accuracy from 93.6% to 94.4%. In contrast, for models that already performed very well such as RF and SVM, the hybrid approach provided minimal or no gains. Listing 3 displays a rule labeling a tumor as `malignant` if cell size uniformity is not 1 and no relevant exceptions arise. Among these exceptions is `ab1`, shown in Listing 3, handling cases where, despite non-uniform cell size, other features do not indicate malignancy. For example, `patient33` was initially misclassified as "benign" by the ML model but was correctly classified as "malignant" by the hybrid model, as `cell_size_uniformity` was equal to 5 (and thus not 1), and additionally, exceptions `ab1` and `ab2` did not hold.

In this dataset, all correct predictions by the ML model were confirmed by the hybrid approach. Consequently, the hybrid approach provided consistent or improved results.

5.1.4 Ecoli

The hybrid model significantly improved SVM accuracy from 57.1% to 87.5%. Listing 4 describes a rule labeling a sample as `cp` if its sequence name (`sn`) is not "FECR" and the attribute `alm1` is less than or equal to 0.38. Listing 4 also defines an exception that checks various attributes (`gvh`, `mcg`) to invalidate the rule when these features are atypical.

For example, `patient48` was classified as "negative" by the ML model. As the first rule in Listing 4 applied, hence, the hybrid model corrected the classification to "positive".

In several examples, if the ML model and the ASP converged on the same conclusion, the final prediction remained unchanged. On the other hand, the hybrid system can misclassify instances when a rule is activated at the wrong threshold or lacks the proper exception. In one instance, for example, the SVM model accurately predicted "negative", but the hybrid system changed the prediction to "positive".

5.2 Explainability Analysis

In addition to performance improvements, our hybrid approach enhances explainability through interpretable ASP rules, which we examine in this section.

When ML models make correct predictions with high confidence, the hybrid model confirms these predictions while adding transparent explanations. For instance, in the Heart Disease dataset (`patient4`), the SVM model correctly predicted "present" with 0.77 confidence. The hybrid model confirmed this, while explaining that although "absent" rule was partially activated, the critical exceptions were not met.

A key consideration is explainability when the ML classifier and FOLD-R++ disagree. In the Breast Cancer dataset ((`patient22`), the MLP model correctly predicted "malignant" with high confidence (nearly 1.0), whereas the FOLD-R++ incorrectly predicted "benign". The hybrid model maintained the correct ML prediction based on confidence thresholding. The ASP explanation for this instance indicates that although the rule for a malignant classification was partially activated, several critical conditions were not met. Consequently, the rule did not fully trigger a malignant classification. This case highlights a limitation in our approach: while the system correctly maintained the ML prediction due to confidence thresholding, the available ASP explanation would have supported an incorrect prediction. Thus, there is a trade-off between interpretability and accuracy.

Conversely, in the Autism Screening dataset (`patient2`), the SVM model incorrectly predicted "autistic" with 0.01 confidence, while FOLD-R++ correctly predicted "non-autistic". Consequently, the hybrid model overrode the uncertain ML output. The ASP explanation for this instance shows that the rule for a "non-autistic" diagnosis was activated and other condition checks were met, while no exception conditions were triggered. Here, the ASP explanation not only corrected the ML prediction but also provided a clear rationale that could be validated by domain experts.

These examples demonstrate that while our approach generally enhances explainability, explanation reliability depends on rule quality, confidence thresholds, and alignment with domain knowledge. When ML classifiers have high accuracy, the added value of ASP explanations may be limited from a performance perspective, but they still increase interpretability. For lower-accuracy ML models, ASP rules can simultaneously improve performance and provide reliable explanations.

6 Conclusion

This study investigated the integration of interpretable ASP rules derived from the FOLD-R++ algorithm with black-box ML models to improve predictive performance and interpretability in medical classification tasks. A central advantage demonstrated by this method is its ability to integrate ASP-derived rules without modifying the internal mechanisms of the classifiers. As shown in the experiments, this offers a practical way to combine robust ML predictions with explainable rules that can correct low-confidence or erroneous outputs. The study addressed two research questions.

First, regarding whether integrating interpretable ASP rules improves the predictive performance of black-box ML models across various medical datasets, the results demonstrate significant improvements. The hybrid model notably improved the accuracy and F1 scores of several ML classifiers, particularly the SVM, across multiple datasets. For example, in the Autism Screening Dataset, the hybrid model increased the accuracy of the SVM classifier from 72.6% to 94.0% ($p < 10^{-10}$). Similarly, in the Ecoli dataset, the accuracy improved from 57.1% to 87.5% ($p < 0.0001$). This suggests the hybrid model was able to address limitations in certain ML models when dealing with complex or noisy data. In general, the largest improvement where seen on sub-optimal ML classifiers.

The results demonstrate that while the hybrid approach significantly enhances performance for classifiers with lower baseline scores, there are also datasets where standalone FOLD-R++ outperforms both the baseline ML and the hybrid methods. This reflects our core research objective of enhancing ML classifiers, rather than asserting that the hybrid approach exceeds all alternatives. Notably, even when ML models perform better than standalone FOLD-R++, as in the CDK dataset, the hybrid approach can still offer incremental improvements while preserving interpretability.

Second, concerning how the hybrid model improves the interpretability of predictions, the integration of ASP rules provides clear, human-readable explanations for each prediction. The logical rules derived from the FOLD-R++ algorithm align with domain knowledge, and make it easier to understand and trust the predictions. For instance, in the Chronic Kidney Disease dataset, the rule for labeling "ckd" reflects the medical understanding that elevated serum creatinine levels (`sc`) are indicative of kidney dysfunction. The rules improved the accuracy of predictions and provided an explanation aligned with medical knowledge, which improves transparency and trust.

Although the current implementation does generate an ASP-based proof tree for every instance, the primary advantage occurs where the hybrid system actually overrides a low-confidence or misclassified ML prediction. In those cases, the symbolic justification is crucial for conveying why the override applies. In high-confidence cases that the ML model already classifies correctly, the hybrid model typically does not rely on the proof tree to alter the outcome, though an explanation is still recorded. By contrast, if the prediction is already high-confidence and correct, the system does not rely on the proof tree to alter the outcome. If every prediction requires a detailed rationale, one could incorporate additional post-hoc explainers (e.g. SHAP-FOLD [SG19]).

However, while the hybrid model shows promising results, several limitations should be considered. As datasets become larger and more complex, the number of induced ASP rules can increase substantially, which could affect interpretability. Additionally, heavy use of exceptions for capturing specific cases may cause the rules to overfit the training data. The variations in induced rules across different experiments suggest sensitivity of the FOLD-R++ algorithm to data splitting and randomness. Furthermore, the hybrid model did not consistently improve performance across all ML models and datasets, as the benefits for KNN and RF on certain datasets were minimal, indicating the effectiveness of the hybrid approach depends on the characteristics of both the dataset and base ML model.

Finally, running an ASP solver like Clingo introduces additional computational overhead. For larger datasets, this may require further optimizations.

Future work should extend this research across several dimensions. A primary focus will be on enhancing scalability, particularly by addressing the computational overhead of ASP solvers like Clingo on larger datasets and exploring alternative symbolic engines. Broader empirical evaluations on more diverse, larger datasets and applications to new domains will also be pursued, incorporating a wider variety of ML architectures to benchmark against state-of-the-art models. Systematic optimization of the confidence threshold and further investigation into the effects of data transformation will be pursued, including assessing FOLD-R++'s capabilities with less transformed or original data. To address the variability observed in induced rules and improve their robustness, dynamic rule refinement techniques, such as ILP, will be explored. Further research includes human-in-the-loop evaluations with domain experts to validate interpretability and clinical utility, and the investigation of more advanced hybrid architectures.

Acknowledgements

This work is partially supported by the project LogicLM: Combining Logic Programs with Language Model with the number NGF.1609.241.010 of the research programme NGF AiNed XS Europa 2024-1 which is (partly) financed by the Dutch Research Council (NWO).

References

[ADRDS+20] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser et al. (2020): *Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI*. Information Fusion 58, pp. 82–115, doi:10.1016/j.inffus.2019.12.012.

[DVK17] Finale Doshi-Velez & Been Kim (2017): *Towards a rigorous science of interpretable machine learning*. arXiv preprint arXiv:1702.08608, doi:10.48550/arXiv.1702.08608.

[EGL16] Esra Erdem, Michael Gelfond & Nicola Leone (2016): *Applications of answer set programming*. AI Magazine 37(3), pp. 53–68, doi:10.1609/AIMAG.V37I3.2678.

[GGL+19] Artur d'Avila Garcez, Marco Gori, Luis C Lamb, Luciano Serafini, Michael Spranger & Son N Tran (2019): *Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning*. arXiv preprint arXiv:1905.06088, doi:10.48550/arXiv.1905.06088.

[GKKS19] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2019): *Multi-shot ASP solving with clingo*. Theory and Practice of Logic Programming 19(1), pp. 27–82, doi:10.1017/S1471068418000054.

[GMR+18] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti & Dino Pedreschi (2018): *A survey of methods for explaining black box models*. ACM computing surveys (CSUR) 51(5), pp. 1–42, doi:10.1145/3236009.

[KLN25] Markelle Kelly, Rachel Longjohn & Kolby Nottingham (2025): *The UCI Machine Learning Repository*.

[Lun17] Scott Lundberg (2017): *A unified approach to interpreting model predictions*. arXiv preprint arXiv:1705.07874, doi:10.48550/arXiv.1705.07874.

[MDK+18] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester & Luc De Raedt (2018): *Deepproblog: Neural probabilistic logic programming*. Advances in neural information processing systems 31.

[PLV24] Kavitha Palaniappan, Elaine Yan Ting Lin & Silke Vogel (2024): *Global regulatory frameworks for the use of artificial intelligence (AI) in the healthcare services sector*. In: Healthcare, 12, MDPI, p. 562, doi:10.3390/healthcare12050562.

[Rud19] Cynthia Rudin (2019): *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*. Nature machine intelligence 1(5), pp. 206–215, doi:10.1038/S42256-019-0048-X.

[SG19] Farhad Shakerin & Gopal Gupta (2019): *Induction of non-monotonic rules from statistical learning models using high-utility itemset mining*. arXiv preprint arXiv:1905.11226, doi:10.48550/arXiv.1905.11226.

[STD17] Matteo Spiotta, Paolo Terenziani & Daniele Theseider Dupré (2017): *Temporal conformance analysis and explanation of clinical guidelines execution: An answer set programming approach*. IEEE Transactions on Knowledge and Data Engineering 29(11), pp. 2567–2580, doi:10.1109/TKDE.2017.2734084.

[TJMG19] Sana Tonekaboni, Shalmali Joshi, Melissa D McCradden & Anna Goldenberg (2019): *What clinicians want: contextualizing explainable machine learning for clinical end use*. In: Machine learning for healthcare conference, PMLR, pp. 359–380.

[WG22] Huaduo Wang & Gopal Gupta (2022): *FOLD-R++: a scalable toolset for automated inductive learning of default theories from mixed data*. In: *International Symposium on Functional and Logic Programming*, Springer, pp. 224–242, doi:10.1007/978-3-030-99461-7_13.

[YIL23] Zhun Yang, Adam Ishay & Joohyung Lee (2023): *Neurasp: Embracing neural networks into answer set programming*. *arXiv preprint arXiv:2307.07700*, doi:10.48550/arXiv.2307.07700.

A Additional Dataset Results

This appendix provides detailed results and case study discussions for the Chronic Kidney Disease dataset, supplementing the main analysis in Section 5.

A.1 Chronic Kidney Disease (CKD)

For the SVM classifier, the hybrid model improved accuracy from 91.4% to 93.1%. Listing 5 shows the rule which flags elevated serum creatinine levels as indicative of CKD. For example, `patient70` was incorrectly predicted as "not ckd" by the SVM model. The hybrid model corrected this to "ckd" as the `sc` value (1.3) exceeds the threshold.

```
label(X, ckd) :- sc(X, V_sc_0), V_sc_0 > 1.2.
```

Listing 5: ASP Rule for Labeling `ckd` (Chronic Kidney Disease).

In many instances, the ML model and the ASP rules independently converge on the same prediction. In such cases, the hybrid model simply confirmed the ML prediction. As none of the instances were overridden incorrectly by the rules, the hybrid model never worsened the outcomes. Interestingly, the RF classifier achieved perfect accuracy (100%) on this dataset, while the hybrid approach averaged 99.9%. This slight decrease occurs when ASP rules override correct RF predictions made with confidence below the threshold, highlighting a potential trade-off for near-optimal base models.

Correcting Autonomous Driving Object Detection Misclassifications with Automated Commonsense Reasoning

Keegan Kimbrell Wang Tianhao Feng Chen Gopal Gupta
Department of Computer Science
The University of Texas at Dallas
{Keegan.Kimbrell, Tianhao.Wang, Feng.Chen, Gopal.Gupta}@utdallas.edu

Autonomous Vehicle (AV) technology has been heavily researched and sought after, yet there are no SAE Level 5 AVs available today in the marketplace. We contend that over-reliance on machine learning technology is the main reason. Use of automated commonsense reasoning technology, we believe, can help achieve SAE Level 5 autonomy. In this paper, we show how automated commonsense reasoning technology can be deployed in situations where there are not enough data samples available to train a deep learning-based AV model that can handle certain abnormal road scenarios. Specifically, we consider two situations where (i) a traffic signal is malfunctioning at an intersection and (ii) all the cars ahead are slowing down and steering away due to an unexpected obstruction (e.g., animals on the road). We show that in such situations, our commonsense reasoning-based solution accurately detects traffic light colors and obstacles not correctly captured by the AV's perception model. We also provide a pathway for efficiently invoking commonsense reasoning by measuring uncertainty in the computer vision model and using commonsense reasoning to handle uncertain scenarios. We describe our experiments conducted using the CARLA simulator and the results obtained. The main contribution of our research is to show that automated commonsense reasoning effectively corrects AV-based object detection misclassifications and that hybrid models provide an effective pathway to improving AV perception. [1]

1 Introduction

AV systems have made significant progress over the course of the past several years. Due to the nature of vehicles and road scenarios, it is highly critical that AV systems are perfectly accurate and can accommodate for uncertainties in driving situations such as obstacles, pedestrians, and objects not visible to the camera. These systems must also be able to adjust for road layouts and positions that may not have been encountered in training. A single mistake in classification can lead to accidents and casualties. However, an optimal AV system can assist or control many driving aspects, such as sensing, path planning, speed control, navigating, and parking, reducing accidents caused by human error [17]. This ideal is difficult to achieve since AV systems are heavily dependent on deep learning technologies and may misclassify due to uncertain variables on the road. Therefore, we believe that any improvement to the classification of these systems is significant to the development of safe and accurate AV vehicles.

One ideal for an AV system is to imagine it as a perfect human driver—one who is able to reason like a human who does not commit any errors in judgment or perception. This type of system can be considered as making use of Visual Question Answering (VQA) which gives answers to questions about the visual scenario around the vehicle. Such a system would be effective for both explaining decisions made by an autonomous vehicle to a human controller. However, neural network approaches to explaining visual scenarios tend to work by simply matching word descriptions to the data [9]. Similar issues can be found

[1] https://github.com/UTD-Autopilot/common-sense-reasoning-aided-autonomous-driving

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 128–142, doi:10.4204/EPTCS.439.10

in Large Language Models which can hallucinate, or generate irrelevant information, when answering questions. These systems heavily rely on matching a response to the query and lack true understanding of the context of the scenario. In a safety critical system, we must avoid making mistakes. We believe that commonsense reasoning can be used to properly model human thinking/reasoning [7, 14].

We propose a system that will correct misclassification in the object detection phase of an autonomous vehicle model. The system consists of a base deep learning autonomous vehicle model with a logic program [2] layer on top that is used to improve the output. Commonsense reasoning, realized through logic programming, is used to detect potential inconsistencies in the classifications of the objects in the scenario. The logic program will then output the inconsistency and the explanation for why it believes there to be an issue. We then make adjustments to the output based on the suggested inconsistency to improve the overall accuracy of the computer vision model. In this paper, we will explore the class of possible improvements we can make by observing the collective behavior of vehicles around us to correct misclassification over traffic lights and obstacles on the road.

2 Background

Autonomous Vehicles: Research in autonomous vehicles has experienced massive growth due to the popular development of deep learning and sensor research [25] causing an increase in advanced driver assistance systems (ADAS) in research and commercial vehicles [24]. However, there still exist major gaps in AV research that prevent it from achieving full automation, such as disagreements in optimal perception techniques, algorithms that lack efficiency and accuracy, and the inability to handle difficult road conditions or inclement weather [25]. Here we outline some of the formal definitions for autonomous vehicles that we will use to motivate the rest of the paper. SAE International defines the SAE Levels of Driving Automation into six overall levels [4] ranging from no driving automation to full driving automation, with levels 3-5 being defined as automated driving systems (ADS). These higher levels of automation have been heavily explored in recent AV research, though no SAE Level 5 vehicles have been realized. One of the issues preventing full automation is imperfect perception and sensing technologies, but another major issue lies in the higher-level aspects of driving. Autonomous vehicle technology must deal with four major issues: (i) environment perception and modeling, (ii) localization and map building, (iii) path planning and decision-making, and (iv) motion control. We relate these aspects to the two types of systematic thinking—System 1 and System 2—proposed by Kahneman [10]. System 1 thinking is fast, intuitive, and reflexive, while System 2 thinking is slow and deliberative. Deep learning and computer vision techniques are a very effective and explored approach to handling the first two aspects of autonomous vehicles, as they are heavily based on pattern matching and System 1 thinking. However, the last two aspects require high level reasoning and are more in line with System 2 thinking. Many approaches to AV development consist solely of deep learning techniques for all four aspects, but there is a lot of potential in a hybrid solution that integrates vision and semantics [21, 13]. To this end, we motivate a hybrid approach using deep learning perception and commonsense reasoning. In this work, we will be simulating AV scenarios using the CARLA (Car Learning to Act) simulator [5]. CARLA is an open platform that allows control over sensor suites and environmental factors. In this paper, we will refer to the target vehicle representing the human driver as the *ego* vehicle and the other vehicle objects as *NPC* (non-player character) vehicles. Furthermore, we define the two types of uncertainty that we will be using when making predictions about objects within the CARLA scenarios: Aleatoric uncertainty, which describes general misclassification uncertainty due to randomness in the model, and epistemic uncertainty, which is uncertainty caused by objects that are out of distribution, or not encoun-

tered during training, for the object detection model [8]. We also define birds-eye-view (BEV) semantic segmentation, which is a technique used in autonomous driving to create top-down image representations that segments the image into different categories [18]. An example of this can be seen in Figure 2.

Commonsense Reasoning: To achieve an ideal SAE level 5 autonomous vehicle, the system must be able to comprehend information and reason as a human would. This approach makes use of *commonsense reasoning* to emulate how a human would approach driving. This reasoning is performed over *commonsense knowledge* that we humans possess. Commonsense reasoning is able to handle uncertainties in our perception as well as fill in the gaps. Our approach relies on the observation that humans drivers use *perception* to detect objects (roads, pedestrians, other cars, curbs, traffic lights, lane markings, etc.) in the scene and then use (commonsense) *reasoning* to make driving decisions. Commonsense reasoning can be largely modeled using default rules with exceptions [7]. This represents the human way of thinking about the world around us. For example, a default rule about driving could be *"Drivers will typically stop at a red light"*. However, human knowledge will contain exceptions for those default rules, such as *"Irresponsible drivers don't always stop at red lights"*. This allows commonsense reasoning to handle uncertainty by creating a conclusion with given information and then updating that conclusion based on new incoming information. In the following sections, we give a high level description of how commonsense reasoning—that emulates human behaviour—can be applied to two different road scenarios.

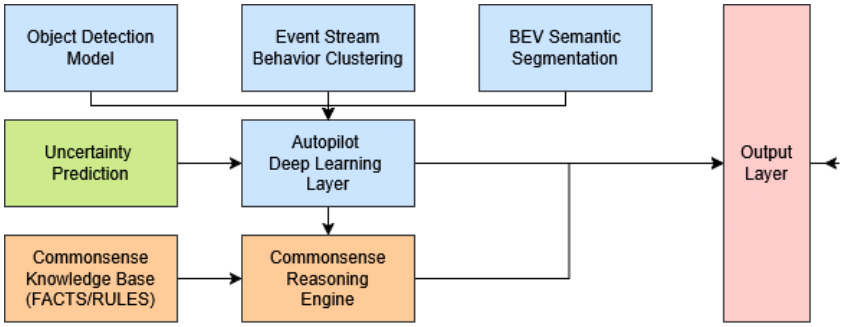


Figure 1: General Procedure Architecture

3 Motivation

While deep learning-based vision and control has seen success in AV technology, a hybrid system brings potential improvements in explainability, human-centred design, and compliance with human-defined industry standards [22]. We motivate this paper by defining the differences between human behavior and commonplace AV behavior. Deep learning is a powerful technology, especially for computer vision. However, due to external factors such as noise, camera or sensor degradation, inclement weather, unusual road scenarios, etc., can cause misclassifications in perception. Perception can also fail in training, and due to the rare nature of accidents, it is possible that an AV does not have sufficient experience with a specific scenario. Due to the black box nature of such systems, such faults in the training may only be noticed after the scenario is encountered, often leading to an accident. In a safety-critical system, such accidents are completely intolerable. It is better to create a set of human-defined commonsense rules that protect the AV when it gets into trouble due to such factors. We take inspiration from the human approach to missing information on the road to write these rules. Like an AV system, it is possible for a

human driver to misinterpret information on the road, such as when they fail to see an obstacle or when they misclassify the color of a stoplight. However, for a human driver, it is possible to use the existing context to reason over the scenario and still make a safe decision. In this paper, we represent how humans look at nearby drivers to better understand a driving scenario. For example, a human driver understands that by default, vehicles are supposed to stop at a stop sign. However, if a human driver notices that a group of vehicles ahead aren't stopping for something that looks like an incoming stop sign, then the stop sign was likely misclassified by them. This ability to reason over incorrect information allows human drivers to avoid potential accidents. We can use this commonsense reasoning approach to correct missing or incorrect information in the perception system in an explainable manner.

4 Methodology

4.1 General Procedure:

Our approach has five major steps. Our goal is to represent perception information as knowledge, and give feedback to the AV by reasoning over that knowledge. The general architecture of our system is shown in figure 1.

Step 1: Use the Autopilot deep learning model to collect visual information about the surrounding area. This includes general object detection, event stream prediction for nearby vehicles, behavior clustering over the predicted event streams, BEV semantic segmentation, and *epistemic* or *aleatoric* uncertainty predictions in our secondary approach.

Step 2: Define commonsense reasoning facts/rules to reason about the possible explanations for each specific scenario (e.g., traffic light, obstacle, stop signs).

Step 3: Perform commonsense reasoning about the relations between the collective behaviors of all vehicles and non-vehicle objects predicted by the Autopilot deep learning model and identify the violated rules. Our experiments consider two approaches to this step: allowing the commonsense reasoning to be active the whole time the system is running and using uncertainty to invoke the commonsense reasoning only on frames where it is needed.

Step 4: Assume that detected collective behaviors are true to connect violated rules to possible false predictions. For example, if the collective behavior is that vehicles will change to a different lane when they are near a region of a specific lane and far away from intersections, there is only one explanation: there is an obstacle in that region. However, no obstacles were detected by the deep learning model in that region. Then this explanation is violated, so we can predict that there should be an obstacle in that region, which is a misclassification of the deep learning model.

Step 5: Make adjustments in the output layer to the non-vehicle objects predicted by the deep learning model to make the scenario consistent again and output the final result.

4.2 Perception and Behavior Modeling

Sensors commonly used in autonomous vehicles, such as cameras, LiDAR, radar, IMU, and GNSS, only produce low-level information, which is difficult to directly use in a commonsense reasoning system. In order to obtain high-level information that is more helpful to the models, we used several deep learning models. Birds-eye-view (BEV) semantic segmentation is a cutting-edge technique widely used in autonomous driving. Given the image captured from surrounding cameras installed on an autonomous

vehicle, BEV semantic segmentation model transforms those images into a top-down perspective and classifies them into predefined categories, such as roads, vehicles, pedestrians, and other objects. Compared to camera space semantic segmentation, the BEV semantic segmentation offers a clear and comprehensive view of the surroundings, enabling better decision-making processes. In this paper, we consider Lift-splat-shoot [18] as the BEV semantic segmentation model. The model takes images captured by six cameras installed around the vehicle as input and outputs a BEV semantic segmentation containing four classes: vehicles, drivable area, lane markings, and others. In CARLA, we mounted six cameras on top of the ego vehicle at six different angles: -120, -60, 0, 60, 120, and 180. Each camera has a field-of-view (FOV) of 90 degrees. The birds-eye-view image is a 500×500 RGB image, representing a $100m \times 100m$ range, where the ego vehicle is located at the center of the image. The ground truth birds-eye-view image for training is created in Carla by mounting a camera 50 meters above the ego vehicle facing straight down, with a FOV of 90 degrees.

Traffic Light Detection: Traffic light information is very important for decision-making at intersections. Many traffic light poles contain multiple groups of traffic lights, each group corresponding to different directions and lanes. Depending on which lane our ego vehicle is in and which direction we want to go, we may need to look at different groups of lights. While detecting the state of the lights is relatively easy, understanding the meaning of the light can be challenging. In this paper, we consider a simpler setting; we only detect the traffic light that is impacting the ego vehicle. In other words, we're trying to figure out whether we should go or stop at the traffic light. For this task, we trained a CNN that takes the front camera image to do a binary classification; the positive class means that the ego vehicle is being impacted by a red traffic light and thus it should stop, and the negative class means the vehicle is not being impacted by a red traffic light and it can go. Carla provides an API *"Vehicle.get_traffic_light()"* which will return the state of the traffic light that is affecting the vehicle. When multiple vehicles are being affected by a traffic light, only the first vehicle is able to get the correct state, so we implemented a function to fetch the traffic light state of the first vehicle in a lane, and we used the return value of this function as the ground truth.

Behavior Detection: In order to detect collective behavior, we first need to detect the individual behavior of the vehicles. We defined 6 categories of events: *ChangeLaneLeft*, *ChangeLaneRight*, *Straight*, *TurnLeft*, *TurnRight*, and *LaneFollow*. In these six events, *Straight*, *TurnLeft*, and *TurnRight* can only happen at an intersection. The behavior prediction model is a classification model based on "resnext50_32x4d". For each obstacle vehicle in the BEV range, we translate and rotate the BEV image so the obstacle vehicle is located at the center of the image and heading up. Then we predict the behavior of the vehicle. Ground truth behavior is obtained by looking at the future trajectory of the vehicle. For example, if the vehicle is turning right at an intersection, we mark the behavior of the vehicle to *TurnRight* from when it starts to turn until it finishes. With the knowledge of the future trajectory, we wrote a few simple pattern-matching rules to obtain the ground truth behavior. The model was then trained based on the collected ground truth data.

Behavior Clustering: Within a time window, if multiple vehicles close to each other perform the same behavior, we treat this group as a cluster. With the individual behavior known, we can detect behavior clustering with classical clustering algorithms. To obtain behavior clusters, we consider a 20-second sliding time window. In the time window, for each category of behaviors, we use DBSCAN on the location of where the behavior starts to try to form a cluster. DBSCAN eps is set to 2.7, and the minimum cluster size is set to 2. If a cluster is found, we save the information about the cluster for use in reasoning.

Uncertainty Prediction: The uncertainty of the deep learning models is evaluated with evidential deep learning [20]. Evidential deep learning estimates the *aleatoric* and *epistemic* uncertainty of an underlying

model by replacing its output class probability with a Dirichlet distribution. Given a deep learning based classification model $f(X;\theta) \rightarrow Y$, where X is the input, θ is the model's parameter and $Y \in C$ is the class label, we replace the final activation with a ReLU function to derive the concentration parameters $\alpha = \sigma_{\text{ReLU}}(f(X;\theta))$ of the Dirichlet distribution $Dir(\alpha)$. Denote the expected probability for class c as $p_c = \frac{\alpha_c}{\sum_{i=1}^{C} \alpha_i}$, we minimize the following objective:

$$\mathscr{L}(\theta) = \sum_{i=1}^{N}(\sum_{j=1}^{C}(y_{ij} - p_{ij}) + \frac{p_{ij}(1-p_{ij})}{(\sum_{j=1}^{C}\alpha_{ij})+1}) + \lambda_t \sum_{i=1}^{N} KL(Dir(\mathbf{p_i}|\alpha_\mathbf{i})||Dir(\mathbf{p_i}|(1,...,1)) \quad (1)$$

where $\lambda_t = min(1.0,t)$ is the annealing coefficient, t increase over training, $Dir(\mathbf{p_i}|(1,...,1))$ is the uniform Dirichlet distribution and $KL(\cdot)$ is the KL divergence. The aleatoric uncertainty and epistemic uncertainty can then be calculated with:

$$u^{\text{alea}} = -\max_c p_c, \quad u^{\text{epis}} = \frac{C}{\sum_{i=1}^{C}\alpha_i} \quad (2)$$

We adopted the traffic light detection model and BEV semantic segmentation model and calculated both aleatoric and epistemic uncertainty. Traffic light detection uncertainty is a numerical value, and BEV uncertainty is obtained at pixel level.

4.3 Commonsense Reasoning Integration

The main contribution proposed by this paper is using a commonsense reasoning layer to perform consistency checking over the deep learning model. The commonsense reasoning models commonly held information about how nearby vehicles behave when presented with certain traffic scenarios. For the dataset used in this experiment, CARLA, we format the data so that each frame contains information objects in the scene, such as vehicles, traffic lights, lanes, and intersections, as well as predictions over actions and action clusters for vehicles within the scene.

Traffic Lights: The deep learning model passes relevant information about nearby intersections, the color of incoming traffic lights, the current location of objects in the scene, and the event stream for vehicles in the scene to the commonsense reasoning module. If the color of the traffic light is classified incorrectly, the commonsense model will detect the inconsistency and output the issue and why it came to that conclusion. For traffic lights, it creates the following representations:

```
property(vehicle, Frame, Object_id, Action, VelocityX, VelocityY,
    Rotation, Coordinate1X, Coordinate1Y, Coordinate2X, Coordinate2Y).
property(intersection, Frame, Object_id, Coordinate1X, Coordinate1Y,
    Coordinate2X, Coordinate2Y),vehicles(Frame, Vehicles).
```

These facts encapsulate the information about vehicle and intersection objects and list the frames they are contained in. The code then makes a call to the main traffic light inconsistency checking rule *consistent_intersection(Frame)*.

```
red_traffic_light(Frame):- property(intersection, Frame, _, _, _, _, _),
    stopped_vehicle_in_front(Frame), ...
red_traffic_light(Frame):- property(intersection, Frame, _, _, _, _, _),
    collective_up_straight(Frame), \+ ego_collective(Frame), ...
```

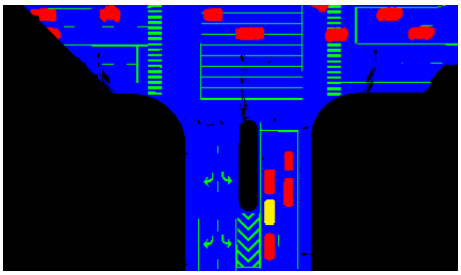


Figure 2: A birds eye view semantic segmentation of a traffic scenario at an intersection..

This main rule determines the output of the logic side of the hybrid model for detecting red traffic lights. *Frame* is the number value of the frame where the inconsistency is being checked. The *red_traffic_light* rule has several variations that observe various possible collective behavior scenarios along with their relation to the ego vehicle to determine the state of the light. To supplement the main collective behavior rule, we introduce a rule for detecting the action of the vehicle in front of the ego vehicle closest to the traffic light.

```
stopped_vehicle_in_front(Frame):-
    driver_location(Frame,CoordinateX,CoordinateY),
    driver_rotation(Frame, Rotation),...
    property(vehicle, Frame, First_Vehicle_ID, _, VelocityX,
    VelocityY, Rotation2, Coordinate2X, Coordinate2Y, _, _),
    property(traffic_light, Frame, _, First_Vehicle_ID, ...), ...
```

This new rule detects the presence of a vehicle in front of the ego vehicle at a given traffic light. We perform calculations to detect the movement status of this front-most vehicle. Similar to the concept explored in Section 3, this models human behavior. In Figure 2, for the given road scenario, a human can quickly identify whether or not there is a traffic light at this intersection. The blue space is the driveable surface, the red boxes are vehicles, and the ego vehicle is the yellow box in the center of the image. In this scenario, we can see a group of vehicles moving left across the intersection. Because of this, we can tell that a traffic light must be at this stoplight and that it is red for the ego vehicle. This is because there are too many vehicles moving across for it to be a probable traffic violation.

Obstacles: Another form of inconsistency that is possible in a scenario is undetected obstacles on the road. This is potentially very dangerous and can happen to a human driver who is not completely observant of the road in front of them. For this scenario, we used the cluster event stream changes given by the deep learning model. This knowledge is encapsulated as the logic program fact *change_action_cluster*. This fact represents when a group of vehicles change their actions within a certain area. The cluster action change also contains the frame window in which the cluster occurs as well as the location of the cluster. The main rule that detects this inconsistency, *collective_obstacle_detection(Frame)*, determines if the current frame is within the given frames of the cluster action change. *Frame* is the frame number where the inconsistency is being checked. *FrameStart* and *FrameEnd* are the number values of the frame when the cluster begins and when it stops. *Action* is the new action for the vehicles being taken within the cluster. Finally, *Ego* determines whether or not the ego vehicle is within this cluster.

```
change_action_cluster(FrameStart, FrameEnd, Action, Ego
Coordinate1X, Coordinate1Y, Coordinate2X, Coordinate2Y).
explainable_lane_change(Frame):-
```

```
change_action_cluster(FrameStart, FrameEnd, Action, Ego, ...),
Frame >= FrameStart, Frame =< FrameEnd,
\+abnormal_obstacle(FrameStart, Action),
\+property(intersection, Frame, Object_id, C1X, C1Y, C2X, C2Y).
```

If a cluster lane change is found around the ego vehicle, this rule then checks if there are any ways to explain it. For example, an intersection may be nearby, which explains why many vehicles would change right within the same area. If no exception can be determined, then the logic determines that it is an inconsistent scenario and that there may be an undetected obstacle ahead. This version of the rule states that a cluster change of vehicles around an intersection is not particularly notable. The existence of the intersection explains the scenario, and so the scenario is determined to be consistent.

5 Experiments

Experimental Setup: For each scenario, we compute the results of deep learning model augmented with commonsense reasoning (referred to as *hybrid* in Tables 1-4) with the output of the baseline deep learning-only model (referred to as *baseline* in Tables 1-4). The final metric is generated by comparing the final result in each case with the ground truth data. For traffic light scenarios, we evaluate the color of the traffic lights, and for obstacle scenarios, we evaluate the detection of the obstacle. We perform evaluations over data collected from the CARLA simulator. The CARLA simulator data is collected using the Autopilot deep learning model. This data is taken from several recordings from CARLA Towns 1-4 at different vehicle densities varying between 100-200 vehicles in each recording. The data collected includes object detections, collective behavior clustering, BEV semantic segmentation, and uncertainty predictions. We use simulation recordings to evaluate our approach for traffic light and obstruction scenarios. In each scenario, we compare the accuracy, precision, recall, and F-measure of the hybrid model with the baseline model. Additionally, we consider two types of commonsense reasoning layers: the fully active commonsense reasoning layer and the uncertainty-invoked commonsense reasoning layer. These layers are constructed through two components: the external commonsense knowledge base representing facts and rules held by a human driver, and the knowledge generated by the Autopilot model, which represents facts about the visual scenario.

Results and Evaluations: The results of our system over the CARLA simulator data for traffic light scenarios are shown in Table 1. As we can see, the hybrid results significantly improve all the metrics compared to the baseline. Note that while the logic layer depends on the deep learning system for perception, the evaluation of our target scenarios and objects is separate. This means that for the traffic light scenario, the logic layer uses all of the data provided by the deep learning model except for the state of the traffic light. As mentioned, we evaluate the effectiveness of both our *baseline* detection model that uses only deep learning and then we evaluate it again when *combined* with the commonsense reasoning layer. For the logic model, we only consider frames in which the logic applies, which consist of frames in which there is a nearby intersection and at least one nearby collective behavior. We also don't consider the first few frames after a light changes from red to green to give time for a collective behavior to be detected. For the baseline detection model and the hybrid model, we evaluate them over all frames in the dataset. In the hybrid model, when there is a discrepancy between the traffic light detection model and the logic model, we treat this as an inconsistency. In the system proposed in this paper, we favor the output from the logic system. In future systems, alternate actions could also be taken, such as providing a warning with an explanation, suggesting alternate action, or requesting the perception model to reevaluate a target area.

5.1 Fully Active Commonsense Reasoning Layer

The CARLA datasets used are recordings taken over CARLA Towns 1-4 using multiple vehicle densities. We currently evaluate a total of 100 and 200 NPC vehicles We have also introduced inclement weather into these scenarios. These factors affect the image data and the accuracy of the traffic light detection model. As one can see in the table, the logical layer evaluates at around 70-95% for all metrics provided, while the baseline model has significantly worse performance. This provides a large increase in accuracy for the overall model when the traffic light detection model fails to achieve good accuracy. For example, in the Town 1 100 NPCs dataset, the weather condition was rainy. There were several frames in which the traffic light detection model failed to detect a red light, but this inconsistency was caught by our logic system. This allows the hybrid system to achieve an accuracy increase of anywhere from 5-56% over the baseline detection model in all the recordings, depending on how frequently the baseline failed on scenarios that were captured by the commonsense. In Table 2 we see the results of the scenario in which a stopped vehicle is obstructing traffic. Once again the results for the hybrid approach are significantly better than those for the baseline. In fact, accuracy is 100% in all cases for the hybrid approach. These recordings are primarily taken from CARLA Town 3, with varying NPC densities. More dense scenarios were considered since we rely on detecting collective behaviors, which occur more frequently in more vehicle-dense environments.

Table 1: Results of commonsense reasoning, baseline deep learning, and hybrid models for traffic lights over Towns 1-4 at 100 and 200 NPC densities.

Fully Active Traffic Lights	Accuracy	Precision	Recall	F-Score
Town 1 100 Baseline	.4789	.6578	.1445	.2369
Town 1 100 Hybrid	**.8546**	.9297	**.9942**	**.9608**
Town 1 200 Baseline	.7634	.5000	.4091	.4500
Town 1 200 Hybrid	.8387	.6400	.7272	.6809
Town 2 100 Baseline	.2446	.6757	.0535	.0992
Town 2 100 Hybrid	.6672	.8825	.6595	.7549
Town 2 200 Baseline	.5071	.5161	.1524	.2353
Town 2 200 Hybrid	.8418	.7939	.9429	.8646
Town 3 100 Baseline	.4083	.3333	.0491	.0855
Town 3 100 Hybrid	.5502	.5954	.6319	.6131
Town 3 200 Baseline	.2573	.8235	.1014	.1806
Town 3 200 Hybrid	**.8129**	.9206	**.8406**	**.8788**
Town 4 100 Baseline	.3515	.6774	.0769	.1381
Town 4 100 Hybrid	.4059	.7088	.2051	.3181
Town 4 200 Baseline	.6272	.3035	.1197	.1717
Town 4 200 Hybrid	.6931	.5412	.3239	.4053

In these recordings, there was an obstruction at each frame, and we evaluated whether or not the baseline model could identify the stopped vehicle. We also determined if the logic model was able to detect an obstruction by noticing a collective lane change when there were no nearby intersections. Like the traffic light scenario, we only evaluate the commonsense reasoning in frames where the reasoning applies. In these particular recordings, there was always a collective lane change, so the commonsense reasoning easily achieves 100% accuracy. The baseline model naturally struggles with identifying an obstruction on the road, since it relies on the obstruction being in camera view. It had varying performance depending on whether the obstruction was visible or being blocked by nearby traffic. In all scenarios, however, the hybrid model is able to reference the commonsense reasoning to achieve 100% accuracy.

Table 2: Results of the commonsense reasoning model for obstructions for the logic, baseline, and hybrid models for recordings over Town 3. Dense means a larger number of NPCs (around 20) in the recording while Mid means fewer (around 10).

Fully Active Obstructions	Accuracy	Precision	Recall	F-Score
Town 3.0 Dense Baseline	.4943	1	.4943	.6615
Town 3.0 Dense Hybrid	1	1	1	1
Town 3.1 Dense Baseline	.93	1	.93	.9636
Town 3.1 Dense Hybrid	1	1	1	1
Town 3.2 Dense Baseline	.392	1	.392	.563
Town 3.2 Dense Hybrid	1	1	1	1
Town 3.0 Mid Baseline	.2284	1	.2284	.3718
Town 3.0 Mid Hybrid	1	1	1	1

5.2 Uncertainty-Invoked Commonsense Reasoning Layer

In the following experiment, we consider the same two traffic scenarios. However, we use our uncertainty prediction models to generate an uncertainty value for the BEV semantic segmentations and the camera inputs for each frame in our dataset. When the uncertainty exceeds a certain threshold, we invoke the commonsense reasoning layer to help handle the uncertainty. This is a more efficient method for allowing the AV model to deal with difficult scenarios without requiring the commonsense layer to be active at all times. In this experiment, we apply aleatoric uncertainty towards the traffic light scenarios and epistemic uncertainty towards the obstacle scenario. For the traffic light scenarios, noise has been applied to the traffic light object detection model, causing there to be uncertainty in the quality of the traffic light classification. We took a percentage of the average uncertainty across all frames of the dataset and set it as the threshold. If an individual frame exceeded this value, then the commonsense layer is activated that searches for nearby cluster behaviors to determine the status of the light. The results of this experiment can be seen in Table 3, which shows the evaluation metrics for identifying red traffic lights across four different recordings of four different towns in CARLA. The AV system on its own had decent accuracy and maintained a very high precision since it had very few false positives. However, it has a very low recall, meaning that it failed to identify several red traffic lights. *The combined uncertainty/commonsense layer showed massive improvements to the results of the model*. Frames that were labeled incorrectly by the baseline model often had higher aleatoric uncertainty, allowing the commonsense reasoning layer to make corrections. This severely reduced the false negative rate of the system overall. Furthermore, the uncertainty-invoked commonsense reasoning layer ran on much fewer frames than the fully active commonsense reasoning layer. Despite this, it still performed well as an optimizer for the AV system.

For the obstacle detection scenario, we changed the obstacles from stopped vehicles to animals in the road. These animals were not seen in the training set, meaning that the AV system evaluated frames that contained animals with a high epistemic uncertainty. The uncertainty-invoked commonsense reasoning layer then used this value to identify animals on the road. In this experiment, we evaluated the system's ability to detect animal objects directly in front of the ego vehicle. The animal classification was completely out-of-distribution for the baseline model, meaning that it never captured any of the animals obstacles, but it would generate a high epistemic uncertainty for frames where animals were in front of the ego vehicle. The baseline model has higher default accuracy compared to the previous obstacle experiment since the task is slightly different. In this experiment, there are very few positive labels. This emulates a real-world scenario in which an AV system will only encounter certain out-of-distribution objects very rarely, meaning that is typically accurate in the general case but will struggle handling unfamiliar objects. The uncertainty-invoked commonsense layer for this experiment ran on significantly fewer frames than the fully active commonsense layer since there were very few positive

labels and, therefore, very few frames with high epistemic uncertainty. We also used a static threshold in this experiment as opposed to the dynamic threshold used in the traffic light experiment. This allows the commonsense layer to be extremely efficient since it is only expected to run when a very rare scenario occurs. Table 4 shows recordings from two different towns where the ego vehicle encountered animals and failed to identify them. The table shows that the logic model managed to create a small improvement over the baseline model by slightly improving the true positive rate.

Table 3: Results of the baseline deep learning traffic light detection model and the uncertainty-invoked commonsense reasoning

Uncertainty-Invoked Traffic Lights	Accuracy	Precision	Recall	F-Score
Town 1 Baseline	.6444	.9388	.6133	.7419
Town 1 Hybrid	**.9556**	.9610	**.9867**	**.9737**
Town 2 Baseline	.6136	.9782	.5769	.7258
Town 2 Hybrid	.8295	.9846	.8205	.8951
Town 3 Baseline	.6172	.9565	.4230	.5867
Town 3 Hybrid	.8395	.9756	.7692	.8602
Town 5 Baseline	.7931	1	.7882	.8816
Town 5 Hybrid	.9425	1	.9412	.9697

Table 4: Results of the baseline deep learning object detection model and the uncertainty-invoked commonsense reasoning

Uncertainty-Invoked Obstacles	Accuracy	Precision	Recall	F-Score
Town 3 Baseline	.9300	N/A	0	N/A
Town 3 Hybrid	.9495	1	.2857	.4440
Town 4 Baseline	.9400	N/A	0	N/A
Town 4 Hybrid	.9597	1	.3333	.5000

The results from the experiments reveal that applying a feedback layer based on commonsense reasoning is an effective method for handling misclassifications in our perception model. In every experiment for the traffic light scenario, we saw an improvement from the baseline perception model to the hybrid model. The effect of external factors on the baseline perception model, such as inclement weather conditions in this case, can be heavily mitigated. A limitation of this experiment is the imperfect logic. In our experiment, the accuracy of the logic model was around 95% and reached as low as 70%. But despite this, we can still see an increase in performance from the baseline model to the hybrid model. Furthermore, the hybrid model in Table 4 doesn't achieve perfect accuracy like its fully active counterpart. This is due to imperfections in the uncertainty prediction model, causing the logic to not be invoked in every positive label. Despite this, this approach will still always show an improvement, as seen in the table, since it still corrects several objects. With a perfect uncertainty prediction model, it is possible to achieve the same level of accuracy as the fully active model. This is because the commonsense reasoning module will only provide an adjustment in scenarios that are eligible for the logic. The experiment also demonstrated that uncertainty can be used to invoke commonsense reasoning in a more efficient manner while still improving system accuracy when compared to the baseline.

6 Related Work

As research in autonomous vehicles advances, there is more and more work being done on hybrid AI systems combining deep learning and reasoning that are similar to ours. Our approach features improving the knowledge extraction and visual question-answering aspects of autonomous vehicles using collective behaviors; however, similar work has been done recently. For example, the work done by [11] explores collective awareness in a network of agents. They explore how collective behaviors can be detected in a connected system of autonomous vehicles. These collective behaviors are modeled using Dynamic Bayesian Networks and used to identify abnormal situations on the road. This approach uses a statistical relational approach as opposed to a logical approach to identify abnormal scenarios, but it demonstrates a similar concept in using collective behaviors to handle unusual road scenarios. Both approaches are able to generate information about future states based on current information about our scenario. Our approach, however, more closely models human reasoning and can provide a more readable explanation of the scenario, as we are not relying on conditional distributions to identify abnormalities. We also solely rely on camera data to detect abnormal scenarios, as opposed to needing information from a group of agents. This does come at the cost of scalability. This sort of agent based approach to identifying collective behavior is also explored by [19]. They designed a system that measures Social Value Orientation to predict the behaviors of vehicles on the road. Their approach is similar to ours in that they use the behaviors of vehicles to generate new information about the current road scenario. We agree with their assertion that using the behaviors of other vehicles in an AV system allows it to be more intelligent and socially aware.

Similar work focusing on more logical approaches can be seen in other AV scenarios. [3] created an Inductive Logic Programming (ILP) [16] system designed for traffic sign detection. Their approach is robust against perturbations in the images, but is more focused on combining computer vision techniques with ILP to perform traffic sign detection. They look more into computer vision-based attributes such as shape, color, and text to handle misclassifications, as opposed to our approach, which makes use of information about the traffic scenario around us. A similar approach to the traffic sign detection task using our system would involve observing the behavior of nearby vehicles and determining if it can be used to determine the classification of an uncertain traffic sign. Another work by [12] focuses on using Answer-Set Programming (ASP) to perform continuous monitoring and fault detection for an AV system. Their ASP system is designed to help identify and explain differences between predicted and actual vehicle behavior. They demonstrate that their approach is effective at providing quality assurance and detecting violations within the AV system. Our approach is more directly involved with the sensor data, particularly the camera data, and resolving issues within the perception of the AV system, as opposed to its behavior. While we don't explore adjusting or identifying the behavior of the AV system within this paper, our work can be extended to this task by using corrections in the perception, such as correcting a traffic light from green to red, to help correct the behavior of the AV system.

Research into more logical hybrid systems has also been explored in recent years. [1] propose a symbolic model-based approach to generating logical scenarios for autonomous vehicles. [21, 22] have also explored many models that combine commonsense reasoning with AVs to create more human-centred, explainable, and ethical vehicles. Their work more closely explores general applications of ASP-based hybrid perception systems, while ours demonstrates a specific commonsense-based optimization. The advantage of our approach is that we can apply our technique to existing AV systems since the commonsense reasoning is decoupled from the perception. Another work, called AUTO-DISCERN, by [13] proposes a framework that automates AV decision-making using answer set programming (ASP). In their work the entire decision-making process uses ASP, while our approach uses commonsense reasoning as a

method of *improving* an AV system. In this sense, our approach is more viable for real-world applications for AV systems. With our approach, we can generate a commonsense base for important safety-critical scenarios and always show an improvement in the decisions created by deep learning-based AVs, as opposed to needing to generate and invoke a large commonsense reasoning base that represents all possible driving decisions and scenarios. This is further emphasized by the fact that AUTO-DISCERN was also not evaluated over any simulators, as we did with CARLA, and their evaluations were ad hoc.

There is significant research on quantifying uncertainties in DL models, typically designed for i.i.d. inputs like images and tabular data. Many existing methods leverage the concept of Bayesian model averaging overall multiple forward passes to estimate distributions over weights or multiple models, such as deep ensembles [15] and dropout-based Bayesian neural networks (BNNs) [6]. However, the requirement of excessive memory and computation burden for training and testing makes it impossible for real-time applications. Recently, a new class of models has been proposed based on the concept of Evidential Deep Learning [23]. They allow uncertainty quantification in a forward pass and with a single set of weights by parameterizing the distributions over distributions.

7 Conclusion and Future Work

We assert that an accurate logic model serves as an effective method for improving the accuracy of perception in autonomous vehicle models through consistency checking. This is indeed demonstrated by our experiments. Because the layers are well separated, we can easily adjust the logical layer to maintain high accuracy without touching any black box models that it is layered on top of. We additionally demonstrated that uncertainty can be used to more efficiently invoke commonsense reasoning. As such, our system reveals an adjustable, explainable, and effective way to perform optimizations for autonomous vehicles. Our future research will focus on building a scalable knowledge base and generating rulesets to cover the entire class of possible misclassifications for an autonomous vehicle. We also plan to evaluate the effectiveness of our approach on real-world datasets and systems.

Additionally, a desirable extension to this work is to use Answer-Set Programming (ASP) [7] to create a more sophisticated modeling of the commonsense knowledge explored in this paper. Commonsense reasoning is heavily based on human reasoning techniques such as defaults, constraints, exceptions, abduction, etc., [7] which can be represented more precisely using ASP. Our future research will involve modeling different possible worlds based on missing information in perception and generating more thorough explanations for a misclassification. Our ultimate goal is to build an AV system that layers commonsense reasoning on top of perception to achieve 100% accuracy, resulting in SAE Level 5 autonomy. The research reported in this paper proves that this is indeed possible. Thus, our main contribution is to demonstrate that perception models augmented with commonsense reasoning can be used to improve object classifications. Expanding the class of problems that can be solved using this approach will allow autonomous vehicles to reach higher levels of autonomy.

References

[1] Boutheina Bannour, Julien Niol & Paolo Crisafulli (2021): *Symbolic model-based design and generation of logical scenarios for autonomous vehicles validation.* In: *IEEE Intelligent Vehicles Symp.*, pp. 215–222, doi:10.1109/IV48863.2021.9575528.

[2] Ivan Bratko (2012): *Prolog Programming for Artificial Intelligence (4th ed.).* Addison Wesley.

[3] Zahra Chaghazardi, Saber Fallah & Alireza Tamaddoni-Nezhad (2023): *Explainable and trustworthy traffic sign detection for safe autonomous driving: An inductive logic programming approach*. arXiv preprint arXiv:2309.03215, doi:10.48550/arXiv.2309.03215.

[4] On-Road Automated Driving (ORAD) Committee (2021): *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*. SAE International, doi:10.4271/J3016_202104.

[5] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez & Vladlen Koltun (2017): *CARLA: An open urban driving simulator*. In: Conference on Robot Learning, PMLR, pp. 1–16.

[6] Yarin Gal & Zoubin Ghahramani (2016): *Dropout as a bayesian approximation: Representing model uncertainty in deep learning*. In: International Conference on Machine Learning, PMLR, pp. 1050–1059.

[7] Michael Gelfond & Yulia Kahl (2014): *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach*. Cambridge University Press, doi:10.1017/CBO9781139342124.

[8] Eyke Hüllermeier & Willem Waegeman (2021): *Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods*. Machine Learning 110(3), pp. 457–506, doi:10.1007/S10994-021-05946-3.

[9] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick & Ross Girshick (2017): *Clevr: A diagnostic dataset for compositional language and elementary visual reasoning*. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1988–1997, doi:10.1109/CVPR.2017.215.

[10] Daniel Kahneman (2011): *Thinking, fast and slow*. Farrar, Strauss, and Giroux.

[11] Divya Thekke Kanapram, Fabio Patrone, Pablo Marin-Plaza, Mario Marchese, Eliane L Bodanese, Lucio Marcenaro, David Martín Gómez & Carlo Regazzoni (2020): *Collective awareness for abnormality detection in connected autonomous vehicles*. IEEE Internet of Things Journal 7(5), pp. 3774–3789, doi:10.1109/JIOT.2020.2974680.

[12] Lorenz Klampfl & Franz Wotawa (2024): *Leveraging Answer Set Programming for Continuous Monitoring, Fault Detection, and Explanation of Automated and Autonomous Driving Systems*. In: 35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024), Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 10:1–10:20, doi:10.4230/OASICS.DX.2024.10.

[13] Suraj Kothawade, Vinaya Khandelwal, Kinjal Basu, Huaduo Wang & Gopal Gupta (2021): *AUTODISCERN: autonomous driving using common sense reasoning*. arXiv preprint arXiv:2110.13606, doi:10.48550/arXiv.2110.13606.

[14] Robert A. Kowalski (2011): *Computational Logic and Human Thinking*. Cambridge University Press, doi:10.1017/CBO9780511984747.

[15] Balaji Lakshminarayanan, Alexander Pritzel & Charles Blundell (2017): *Simple and scalable predictive uncertainty estimation using deep ensembles*. In: Advances in Neural Information Processing Systems 30.

[16] Stephen Muggleton (1991): *Inductive logic programming*. New Generation Computing 8(4), pp. 295–318, doi:10.1007/BF03037089.

[17] Darsh Parekh, Nishi Poddar, Aakash Rajpurkar, Manisha Chahal, Neeraj Kumar, Gyanendra Prasad Joshi & Woong Cho (2022): *A review on autonomous vehicles: Progress, methods and challenges*. *Electronics* 11(14), p. 2162, doi:10.3390/electronics11142162.

[18] Jonah Philion & Sanja Fidler (2020): *Lift, Splat, Shoot: Encoding Images from Arbitrary Camera Rigs by Implicitly Unprojecting to 3D*. In: *16th European Conference on Computer Vision (ECCV 2020)*, Springer International Publishing, pp. 194–210, doi:10.1007/978-3-030-58568-6_12.

[19] Wilko Schwarting, Alyssa Pierson, Javier Alonso-Mora, Sertac Karaman & Daniela Rus (2019): *Social behavior for autonomous vehicles*. *Proceedings of the National Academy of Sciences* 116(50), pp. 24972–24978, doi:10.1073/PNAS.1820676116.

[20] Murat Sensoy, Melih Kandemir & Lance M. Kaplan (2018): *Evidential Deep Learning to Quantify Classification Uncertainty*. *arXiv preprint arXiv:1806.01768*, doi:10.48550/arXiv.1806.01768. Available at `https://api.semanticscholar.org/CorpusID:46939880`.

[21] Jakob Suchan, Mehul Bhatt & Srikrishna Varadarajan (2020): *Driven by commonsense*. In: *ECAI 2020*, IOS Press, pp. 2939–2940, doi:10.3233/FAIA200463.

[22] Jakob Suchan, Mehul Bhatt & Srikrishna Varadarajan (2021): *Commonsense visual sensemaking for autonomous driving*. *Artificial Intelligence* 299, p. 103522, doi:10.1016/J.ARTINT.2021.103522.

[23] Dennis Ulmer, Christian Hardmeier & Jes Frellsen (2023): *Prior and posterior networks: A survey on evidential deep learning methods for uncertainty estimation*. *Transactions on Machine Learning Research*.

[24] Jessica Van Brummelen, Marie O'brien, Dominique Gruyer & Homayoun Najjaran (2018): *Autonomous vehicle perception: The technology of today and tomorrow*. *Transportation Research* 89, pp. 384–406, doi:10.1016/j.trc.2018.02.012.

[25] Ekim Yurtsever, Jacob Lambert, Alexander Carballo & Kazuya Takeda (2020): *A Survey of Autonomous Driving: Common Practices and Emerging Technologies*. *IEEE Access* 8, pp. 58443–58469, doi:10.1109/ACCESS.2020.2983149.

Computing Universal Plans for Partially Observable Multi-Agent Routing Using Answer Set Programming

Fengming Zhu Fangzhen Lin
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong SAR, China
fzhuae@connect.ust.hk flin@cs.ust.hk

Multi-agent routing problems have gained significant attention recently due to their wide range of industrial applications, ranging from logistics warehouse automation to indoor service robots. Conventionally, they are modeled as classical planning problems. In this paper, we argue that it can be beneficial to formulate them as universal planning problems, particularly when the agents are autonomous entities and may encounter unforeseen situations. We therefore propose *universal plans*, also known as *policies*, as the solution concept, and implement a system based on *Answer Set Programming* (ASP) to compute them. Given an arbitrary two-dimensional map and a profile of goals for a group of *partially observable* agents, the system translates the problem configuration into *logic programs* and finds a feasible universal plan for each agent, mapping its observations to actions while ensuring that there are no collisions with other agents. We use the system to conduct experiments and obtain findings regarding the types of goal profiles and environments that lead to feasible policies, as well as how feasibility may depend on the agents' sensors. We also demonstrate how users can customize action preferences to compute more efficient policies, even (near-)optimal ones. The code is available at https://github.com/Fernadoo/MAPF_ASP.

1 Introduction

Consider a situation where a group of robots are programmed to navigate themselves between pre-specified task points. The robots are expected to reach their task points as quickly as possible while avoiding collisions with one another. This problem is commonly encountered in automated warehouses and is typically formulated as a *Multi-Agent Path Finding* problem (MAPF), which is mostly solved using centralized algorithms [32, 33]. Although optimally solving MAPF problems is in principle *NP-hard* [40], in practice there are planners that can efficiently find high-quality solutions even for large-scale instances [30, 18, 25, 24].

However, beyond the logistics industry, the use of such mobile robots in household or office scenarios is also of significant commercial value. The latter presents at least the following distinct features: 1) there is typically no central server computing plans and communicating with all the robots in real time; 2) each robot is usually equipped with some sensors and computing resources, although the capabilities may be limited to maintain cost efficiency; 3) the domain can be relatively small compared to industrial warehouses. These inherently different features may necessitate a potentially different solution concept. More specifically, a feasible solution in an MAPF problem is defined as *a set of collision-free paths* (i.e., sequences of actions), one for each agent. If any contingency occurs, such as one of the robots being perturbed from its planned path or simply stopping accidentally, the entire joint plan must be re-computed in a centralized manner. As mentioned, unlike the situation in a warehouse, home service robots typically do not have access to a central controller during real-time execution. Instead, users may use a smartphone APP to synchronize their robots at the beginning of the deployment, and leave them for their own fully

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 143–166, doi:10.4204/EPTCS.439.11

© Fengming Zhu & Fangzhen Lin
This work is licensed under the
Creative Commons Attribution License.

autonomous routing for a quite long period. Therefore, we propose to adopt the solution concept as *a set of functions*, each mapping an individual agent's sensory readings (also known as observations or local states[1]) to their actions. These functions are referred to as universal plans in planning [29], strategies in extensive-form games, and policies in Markov decision processes (MDPs)[2].

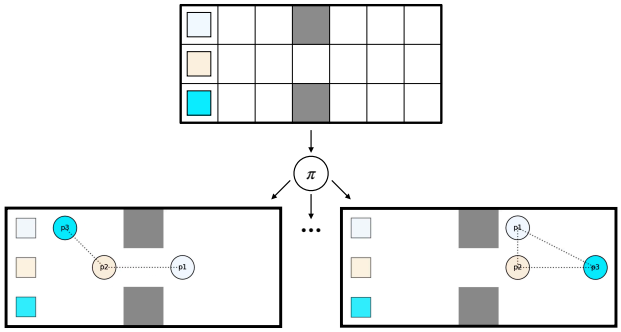


Figure 1: An example (two chambers connected by a tight hallway) that shows how our proposed policy-like solution works. Once computed (taking 167.35s for searching), it will be applicable for all $19 \times 18 \times 17 = 5814$ global states. Colored squares are the goals assigned to sphere-shaped agents in corresponding colors. When two agents are connected by a dotted line, it means they can detect each other.

In this paper, we present a system based on *Answer Set Programming* (ASP) for computing universal plans. We assume multiple agents are moving on a 2D grid map with potential obstacles. Each agent is aware of the map layout and has a designated goal location, but does not know the goals or current locations of the other agents unless they are within its field of view (FoV). Moreover, none of the agents are equipped with communication modules for message sharing. Essentially, our system takes in a problem instance specified by a given map layout and a group of partially observable agents, translates it into ASP encodings, and computes a decentralized policy for each agent. In principle, once a policy profile is found, it works no matter where the agents are placed initially and how they get perturbed during routing. In other words, it can be seen as a paradigm of *centralized search with decentralized execution*. A graphical illustration is provided in Figure 1. With the help of this system, we conduct the following studies, presenting a spectrum ranging from feasibility to optimality.

Feasibility. A policy profile is feasible if all agents can achieve their goals without ever colliding with others. We evaluate the efficiency of our system and report the time cost for computing feasible policies for various configurations. Sometimes there may not exist any feasible policy, and we therefore design specific experiments to investigate how the existence of feasible policies depends on the agents' sensory capabilities.

Action heuristics. To take one step further, one may resort to certain heuristics the improve the quality of the solution policy. Our system provides two ways for the users to specify action heuristics. The first type of heuristic is defined as each individual agent's preferences over actions, e.g., an agent may prefer actions that bring her closer to the goal, regardless of her observations. The second type of heuristic can be interpreted as a set of traffic rules that require agents to adopt a shared protocol when they encounter one another, which is represented as common knowledge among all agents. These action heuristics rule out undesirable policies, and can therefore be an effective strategy for pruning the search space.

[1]We will use these three terms interchangeably. Note that here local states do not refer to the states in any finite automaton.
[2]We will use the terms universal plans and policies interchangeably.

Anytime optimization. Modern ASP solvers enable us to compute optimized solutions in an *anytime* fashion. The longer the program runs, the better the returned solution will be. However, given the high complexity of the overall problem, our experiments show that it is computationally more practical to implement heuristic functions through action restrictions to compute better policies, rather than optimizing from scratch by exhaustive search.

The rest of the paper is organized as follows. Section 2 lists a number of related domains. After that, Section 3 introduces some mathematical notations to illustrate the problem. Next, Section 4 elaborates on the formal encodings to address the problem. Section 5 presents several experiments and findings. Finally, we conclude the paper by discussing potential applications and future directions.

2 Related Work

We list a few related areas that inspire our work, but still, we need to distinguish our work from those.

Compilation-based methods for MAPF. For conventional MAPF problems, apart from leveraging best-first search techniques [30, 18], researchers also investigate methods that compile MAPF instances into formal languages and solve them by declarative programming. Success has been witnessed in formalizations such as SAT [35], CSP [37], and SMT [34]. In particular, we note that one can also formulate such problems in similar or variant settings in nonmonotonic formalisms like Answer Set Programming (ASP) [6, 4, 13, 3]. Nevertheless, the solution concept in these settings is still a sequence of joint actions for a group of agents from their initial positions to goal positions. Our work in this paper aims to compute a universal plan, i.e. a profile of decentralized individual policies, each of which maps an agent's local states (defined under partial observability) to available actions. The policy profile by definition should work for any possible situation.

Universal planning. Another area that closely aligns with ours is known as *universal planning* in single-agent domains. Schoppers [29] first motivated the investigation of universal plans by an example of the well-known *Blocks World*, where a conventional planner may be bothered by a naughty baby who would occasionally smash down the built blocks, and therefore, may need repeated replanning. The idea of universal planning in fact quite overlaps with *contingent planning* or planning under partial observability [5, 41]. Work from those domains also desires policies or conditional plans as the appropriate solution concept. Recent studies, such as [41], have thoroughly investigated the idea of expressing policies by knowledge-based programs, where branching conditions are represented as epistemic formulas [28], even in partially observable domains. This solution concept can also be implemented using ASP in applications related to cognitive robotics [38]. In a comprehensive study, Son et al. [31] have surveyed the maturity and drawbacks of utilizing ASP for planning problems. Therefore, our work can be seen as a multi-agent extension, where each agent is required to execute the distributed plan from its own side.

Dec-POMDPs and MARL. The most related and potentially the most popular domain is *Multi-Agent Reinforcement Learning* (MARL), particularly for cooperative tasks. The underlying model is typically framed as a *Decentralized Partially Observable Markov Decision Process* (Dec-POMDP), which is a Markov chain of states where agents share the same utility function but are not fully aware of their current states [15, 19, 26]. To solve Dec-POMDPs, dynamic programming [14] based on Bellman's principle is typically employed, but it suffers from notorious complexity, as it is generally undecidable to optimally solve infinite horizon Dec-POMDPs [21]. In contrast, our work approaches the problem from the perspective of satisfiability, utilizing efficient constraint propagation and effective heuristics. As a result, our system is capable of solving larger problem instances. There is also a vast body of literature that leverages reinforcement learning techniques to find approximate solutions [20, 7, 27, 39]. However,

learning-based frameworks cannot always guarantee feasible solutions, as they primarily optimize cumulative rewards, which sometimes cannot well capture hard constraints in planning problems. Our system can, to some extent, be viewed as a program that automatically generates "safety rules", but we leave the exploration of how to integrate a logic program into a learning framework for future work.

3 Problem

We consider a group of agents with partial observability moving along a 4-neighbor grid map. Formally, a *configuration* is a 5-tuple $\langle G, N, A, \{r_i\}_{i\in N}, \{g_i\}_{i\in N}\rangle$ defined as follows,

- The 4-neighbor grid map $G = (V, E)$ is an undirected graph, where V is the set of vertices that an agent can be placed at, and E is the set of unit-cost edges.
- N is a set of n agents. Each agent is associated with five unit-cost actions $A = \{\texttt{up}, \texttt{down}, \texttt{left}, \texttt{right}, \texttt{stop}\}$ with their usual meanings. Each agent i is equipped with a sensor of range r_i, enabling it to detect another agent at locations $D_i^v = \{u \in V : d(u,v) \leq r_i\}$ given its own location $v \in V$, where $d : V \times V \mapsto \mathbb{R}$ denotes some distance measure. We also term the detected area as each agent's *field of view* (FoV).
- $\{g_i\}_{i\in N}$ is a goal profile, where g_i is the goal designated to agent i.

The above ingredients consequently define the following notions of *global states* and *local states*,

- The set of global states $S \triangleq V^N$ consists of all possible joint-location tuples of all agents.
- The set of local states $O_i \triangleq \{(v,u) : u \in D_i^v \vee u = \texttt{empty}\}_{v\in V}$ for each agent i consists of its own locations and observations of the others, including the case when it sees no one within its sensor.

A *collision* happens when two agents simultaneously move to the same vertex (i.e., vertex conflict) or traverse the same edge (i.e., edge conflict) [33]. Mathematically, let pos_i^t denote the vertex where agent i is positioned at time step t, then a collision happens between agent i and j if

- $pos_i^t \in V \wedge pos_j^t \in V \wedge pos_i^t = pos_j^t$, or
- $(pos_i^t, pos_j^t) \in E \wedge (pos_i^{t+1}, pos_j^{t+1}) \in E \wedge (pos_i^t, pos_j^t) = (pos_j^{t+1}, pos_i^{t+1})$.

An *instantiation* is to assign an initial profile $\{\alpha_i\}_{i\in N} \in V^N$ to agents, where α_i is the initial position of agent i and $\alpha_i \neq \alpha_j, \forall i \neq j$. This means that no two agents can be placed at the same initial position. We desire a deterministic *policy* $\pi_i : O_i \mapsto A$ for each agent i, which is a mapping from its local states to actions. Note that an intermediate perturbation (or replacement) of agents' locations during routing is equivalent to instantiating them in different initial positions and let them re-route. Given a configuration, a *policy profile* $\{\pi_i\}_{i\in N}$ is *feasible* if the following three conditions are satisfied for all possible instantiations,

1. No collision happens between any pair of agents.
2. Once an agent has reached her goal, she will stop and serve as a fixed obstacle.
3. All agents can successfully reach their goals within finite steps.

A configuration is *solvable* if there exists a feasible policy profile for this configuration.

4 Formal Encoding

In this section, we formalize the problem via the language of *Answer Set Programming* (ASP). We address the problem by first finding feasible solutions and then incorporating optimization procedures. Given that ASP is a prevalent formalism nowadays, we will omit its introduction here. Readers interested in the detailed syntax and stable model semantics are encouraged to refer to [36, 22, 23, 12]. An advantage of encoding this problem into logic programs is that it allows us to easily incorporate human-readable rules to study the behavior of the agents and potentially enhance the quality of the policy.

Some readers may find the enumeration of the encoded rules tedious. Therefore, we have postponed most of the encodings of the facts and domain definitions that are already clear from the previous section to Appendix A for better presentation. Here, we present several key rules that reveal the nature of this problem. To ensure that the policy profile works for all possible situations, we inductively define the reachability of each global state given the policy (`reached/1`). Basically, a global state (`gState/1`) is trivially reachable to itself if all agents have already arrived at their goals (`goal_gState/1`). Inductively, any global state is reachable to the goal state if it can, according to the policy, transit to a successor state that is reachable to goal state, where the transition (`trans/3`) is made by a collection of all agents' actions (`doi/2`) based on their own observed local states (`obsi/2`).

```
reached(S):- goal_gState(S).
reached(S1):- gState(S1), trans(S1,(A1,...,An),S2), reached(S2),
    obsi(Si,ASi), doi(ASi,Ai). % observe and act, for all i
:- gState(S), not reached(S). % rule out infeasible policies
```

Theorem 1 (Correctness)**.** *This ASP implementation (with the complete version attached in Appendix A) is both sound and complete. More specifically, 1) given an answer set of a logic program described above, the policy profile encoded in the answer set by the* `doi/2` *atoms will be a feasible one; 2) conversely, one can construct an answer set of the above logic program from a feasible policy profile.*

The justification is quite straightforward. For one, we have directly encoded those three conditions that define a feasible policy profile. More specifically, 1) in the rule that defines valid global states (`gState/1`), it prohibits any pair of agents from sharing the same location; 2) in the rule that defines valid transitions (`trans/3`), it eliminates any immediate location swaps; and 3) The above reachability rule ensures that every global state can reach the goal state within a finite number of transitions (`reached/1`). Note that the reachable states eventually form a spanning tree rooted at the goal state, with each internal node serving as the root for the states within its subtree.

For optimization, there are several evaluation criteria used in conventional MAPF problems. However, none of them is suitable for our settings, as we require a policy to work in all situations. Thus, we adopt a criterion as minimizing *sum-of-makespan*, which is the sum of the makespans of the execution of the policy under all possible instantiations. Mathematically,

$$\textit{sum-of-makespan}(\{\pi\}_{i\in N}) = \sum_{\{\alpha_i\}_{i\in N}\in V^N:\alpha_i\neq\alpha_j} \textit{makespan}(\{\alpha_i\}_{i\in N},\{\pi\}_{i\in N}).$$

where the metric *makespan* is defined as the number of transitions made from the initial global state to reach the goal global state. Therefore, we can also define *makespan* inductively as follows, denoted as the `dist/2` predicate. A transition between two global states will incur a unit cost.

```
w(0..UPPERBOUND). % pre-defined distance range
```

```
dist(S,0):- goal_gState(S).
dist(S1,C):- not goal_gState(S1), trans(S1,(A1,...,An),S2),
    reached(S1), dist(S2,C-1), w(C),
    obsi(Si,ASi), doi(ASi,Ai). % for all i
#minimize {C, S: dist(S,C)}.
```

Lastly, we emphasize that the computed profile of policies is a collection of *deterministic* and *decentralized* ones, meaning that once deployed on individual agents, each agent will be able to execute the policy independently, and no further intervention from any central controller will be needed.

5 Experiments and Findings

In this section, we categorize our experiments and corresponding findings into three folds.

First, we aim at feasible policies. For the readers' reference, we report the time cost to compute feasible policies for various configurations. Besides, comprehensive experiments are designed to investigate how feasibility depends on agents' sensors. We leave the theoretical characterization for future work.

Second, we restrict the choices of agents' actions by certain heuristics to see if there exists a better yet feasible policy. When the environment is sparse, once we have set appropriate action heuristics, the system may already be efficient enough, as agents normally do not need to sacrifice their own utilities too much due to potentially unnecessary coordinations with others.

Finally, we present two alternatives to optimize the policy. We note that implementing heuristic functions through action restrictions is more effective for computing better policies than optimizing from scratch by exhaustive search.

5.1 Feasibilities

5.1.1 Computational Time

For a rough analysis of the time complexity, each agent is associated with around $\Theta\big(M\binom{K+n-1}{n-1}\big)$ local states, where M denotes the number of all possible positions which an agent can be placed at and K denotes the number of neighboring cells that an agent can detect. As mentioned, the formalized problem is also compatible with the Dec-POMDP formulation. To demonstrate that the capability of our approach excels that of both classical Dec-POMDP solvers [2, 1] and some up-to-date ones [17, 16], we list several benchmarks that are used to evaluate Dec-POMDP solvers in Table 1. One can clearly see that the scales of our tested cases significantly exceed those of the Dec-POMDP benchmarks. For example, for a 4×4 map with three agents, each equipped with sensors of range two (Grid4x4_a3_s2), then each agent will be faced with around 2000 local states, which is an intractable scale for Dec-POMDP solvers.

We report the time cost for computing a feasible policy in Table 2. Programs are solved by *clingo*[3] [11, 10, 9], and all experiments are done on Linux servers with Intel® Xeon® Silver 4210 CPU @ 2.20GHz. *Sensor r* means that the agents are equipped with sensors of range r.[4] More specifically, following our previous notation, a sensor of range r enables an agent to detect any neighboring agent within an L_∞ distance[5] of no more than r. It is observed that different goal locations subtly affect the computational

[3] https://potassco.org/clingo/

[4] For the rest of this paper, we will term *sensor r* and sensors of range r interchangeably.

[5] For all the experiments, we adopt the L_∞ metric as the distance measure, i.e. $dist(p,q) \doteq \max\{|p_x - q_x|, |p_y - q_y|\} \leq r$. One can of course implement any other options, e.g., Manhattan or Euclidian distance.

	#(agents)	#(states)	#(actions/agent)	#(observations/agent)
Dec-POMDP benchmarks [2, 1, 17, 16]				
Tiger	2	2	3	2
Hotel	2	16	3	4
Recycling	2	4	3	2
Grid2x2	2	16	5	2
Grid3x3	2	81	5	9
Our tested cases (selected)				
Grid3x3_a2_s1	2	72	5	48
Grid6x6_a2_s3	2	1260	5	896
Grid4x4_a3_s2	3	3360	5	2196
Grid6x6_a3_s3	3	42840	5	22568

Table 1: Benchmarks for evaluating Dec-POMDP solvers compared to those for our solver.

Map size	2 agents	3 agents	4 agents	5 agents
Sensor 1				
3×3	0.06s	3.23s	6.09min	8.76hr
4×4	0.29s	5.09min	28.40hr	/
5×5	1.61s	40.24min	/	/
6×6	11.69s	6.02hr	/	/
Sensor 2				
3×3	0.06s	3.51s	8.86min	8.47hr
4×4	0.49s	2.22min	15.60hr	/
5×5	2.30s	24.02min	/	/
6×6	7.50s	4.14hr	/	/
Sensor 3				
3×3	0.10s	3.37s	9.00min	8.92hr
4×4	0.39s	2.42min	18.69hr	/
5×5	2.07s	26.71min	/	/
6×6	7.54s	4.46hr	/	/

Table 2: Computational time for various maps, numbers of agents and sensors. *Sensor i* means the agents are equipped with sensors of range *i*. We set the time limit to be 30 hours, and "/" means timeout.

time for a given map, therefore, the results are collected from a single goal profile for each configuration. Note that, unlike learning-based systems, our system solves exact solutions that guarantee feasibility. Once a feasible policy profile is computed, it can be distributed to each agent, and no replanning will be needed regardless of where the agents are positioned on the map.

5.1.2 Existence of Feasible Policies

In addition to scenarios where feasible policies are found, there are cases where no feasible policy exists. Map layouts, sensor ranges, and goal placements all play a significant role in this. We will first provide a necessary condition as follows.

Definition 1. *Given a map G, a goal profile $\{g_i\}_{i\in N}$ is said to be proper, if for any g_i, there is always a path from any non-goal position $v \in V\backslash\{g_j\}_{j\neq i}$ to g_i, without passing through any of $\{g_j\}_{j\neq i}$.*

Map, #agents, obstacle density	sensor 1 #feasible/ #proper/ #total	sensor 2 #feasible/ #proper/ #total
$4\times4, 3$ *agents*, 6%	62.3/ 65.3/100	**65.3**/ **65.3**/100
$4\times4, 3$ *agents*, 12%	33.7/ 41.3/100	**41.3**/ **41.3**/100
$4\times4, 3$ *agents*, 18%	30.0/ 39.0/100	**39.0**/ **39.0**/100
$5\times5, 2$ *agents*, 10%	289.3/289.3/300	289.3/289.3/300
$5\times5, 2$ *agents*, 20%	176.3/197.3/300	**197.3**/**197.3**/300
$5\times5, 2$ *agents*, 30%	105.3/114.0/300	**114.0**/**114.0**/300

Table 3: For each setting, results are averaged over three random seeds. For each generated maps, we randomize 100 goal profiles for 4×4 layouts and 300 goal profiles for 5×5 layouts. 4×4 layouts are tested for three agents and 5×5 layouts are tested for two agents. We set three density levels of obstacles 6%, 12% and 18% for 4×4 layouts, and 10%, 20% and 30% for 5×5 layouts. Those configurations where *sensor 2* is more capable than *sensor 1*, the results are marked in bold fonts.

Proposition 1. *There exists a feasible policy only if the goal profile is proper.*

Proof. This *necessary* condition can be easily justified. Note that agents are not supposed to move any longer once they reach their own goals. If the given goal profile is not proper, then if certain agents are placed right at their goals, there is no chance for the rest to reach their goals. □

Therefore, the number of solvable cases (i.e., configurations under which a feasible policy exists) cannot exceed the number of proper goal profiles. Besides, we also conduct experiments to find out which type of sensors will lead to feasible policies. We start by testing some manually designed maps. It turns out that there may not exist any feasible policy for *sensor 1*, while *sensor 2* is capable enough of ensuring feasible policies for all possible proper goal profile at least in these tested cases. For better presentation, we attach the detailed map layouts and statistics in Appendix B. We additionally test the capability of *sensor 1* versus *sensor 2* under randomly generated maps, with results reported in Table 3, where 4×4 maps are associated with three agents and 5×5 maps are associated with two agents. We report the number of goal profiles that allow feasible policies to exist, the number of proper goal profiles, as well as the total number of all randomly generated goal profiles. As shown in Table 3, all profiles that cannot be solved by *sensor 1* are solvable by *sensor 2* with a success rate of 100%.

An applicable explanation is, if agents are only equipped with *sensor 1*, then they cannot detect each other when they are only two blocks away from each other, i.e., about to collide. Consequently, the rules for eliminating vertex conflicts will in fact propagate more constraints during the solving process, potentially leading to the eventual unsatisfiability of the logic program. More evidence will be shown later in Table 4, where maps in larger sizes and even action heuristics are considered. As it will reveal, even if we force agents to move greedily toward their goals until when they have to coordinate to avoid collisions, sensors of range two are still capable enough of working out feasible policies.

5.2 Action Heuristics

Recall that a policy profile is a collection of individual policies, where each individual policy $\pi_i : O_i \mapsto A$ is a mapping from the agent's local states to available actions. While computing this mapping, it is sometime useful to partition the set of local states into disjoint subsets and impose some restrictions on each subset based on the conditions that the mapping must satisfy. Here, for each agent i, we consider partitioning its local states into two subsets: $O_i = O_i^+ \cup O_i^-$, where $O_i^+ \cap O_i^- = \emptyset$. We then compute

Scenario (map, #agents)	#feasible/#total
Sensor 1	
#1 $(6\times 6,2)$	8/1260
Sensor 2 or 3	
#1 $(6\times 6,2)$	1260/1260
#2 $(6\times 6,2)$	1260/1260
#3 $(5\times 6,2)$	192/870
#3 $(6\times 6,2)$	244/1260
#3 $(6\times 7,2)$	300/1722

Table 4: Feasible policies may or may not exist under certain heuristics. #1, #2, and #3 correspond to scenarios *default action*, *last-minute coordination*, and *myopic preferences*, respectively.

a partial policy for each subset $\pi_i^+ : O_i^+ \mapsto A$ and $\pi_i^- : O_i^- \mapsto A$, and compose the full policy as $\pi_i = \pi_i^+ \cup \pi_i^-$. Users can define their own ways of partition, e.g., let O_i^- be the set of local states where agent i does not detect any other agents, and consequently, let O_i^+ consist of the remaining local states where agent i can detect some others. This will further allow users to specify human-understandable rules for either part (or both) to impose certain properties on the resulting computed policies.

5.2.1 Action Preferences

In this section, we investigate the feasibility of policies based on different choices of O_i^-. To customize a preference ordering over actions, we associate each action with an integral cost in given local state. In principle, any distance measure can be used as a cost heuristic. Here, we present empirical results based on the Manhattan distance from the agent to her goal, ignoring the subsequent actions of other agents. Any action that moves the agent into obstacles or off the map will incur an infinite cost. Note that several actions may have the same cost, i.e., the preference is not strict. Therefore, each agent i with such preferences intends to take actions in a self-interested manner under the local states within O_i^-. We present three scenarios of choosing O_i^-, each of which accordingly defines a different heuristic.

1. *Default action* scenario. Let each O_i^- be the set of local states where agent i have not detected the others in its sensor.
2. *Last-minute coordination* scenario. Let each O_i^- be the set of local states where agent i are more than two blocks away from the others. Note that two agents that are no more than two blocks away may collide into each other if any subsequent coordination is made.
3. *Myopic preference* scenario. Let each $O_i^- = O_i$, i.e. the whole set of local states. This is the most aggressive heuristic that only allows an agent to pick one of the tied actions with the least cost.

Due the limited space, we postpone the rather intuitive encodings for these heuristics to Appendix C. One can notice that, from the first scenario to the third, larger portions of the partition are assigned to each O_i^-, and therefore, stronger constraints are imposed on the policies.

First, we test whether feasible policies still exist under each preference heuristic, with some results reported in Table 4. For each tested configuration, we examine all possible goal profiles, e.g., $36\times 35 = 1260$ proper goal profiles for a 6×6 empty map with two agents. Notice that, for the *default action* and *last-minute coordination* scenarios, *sensors 2* can already solve all configurations. However, it may be too aggressive for the agents to resolve collisions under the *myopic preference* heuristic, as indicated by several failed cases. One can also refer to some detailed elaboration in Appendix E.

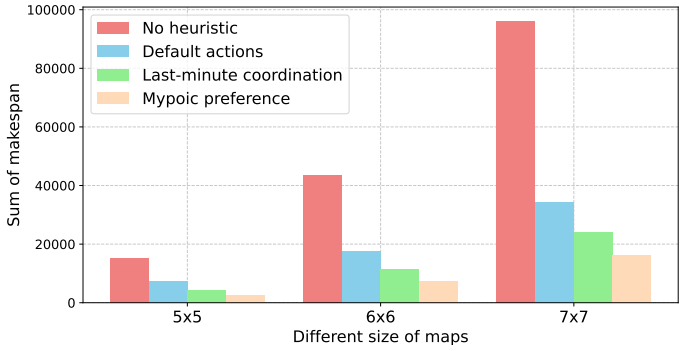


Figure 2: *Sum-of-makespan* evaluated on different maps with different preference heuristics.

Secondly, it is both natural and intriguing to investigate how much the quality of a feasible policy can be improved solely through the use of such action preferences, without any explicit optimization statement included in the program. To this end, we evaluate the aforementioned metric *sum-of-makespan* for the *first* feasible policy found by the ASP solver across different maps for each of the scenarios mentioned above. As shown in Figure 2, significant improvements in the computed feasible policies can be clearly observed, if one exists. An insightful takeaway is that, for one, users may want to design certain single-agent heuristics over O_i^- to effectively prune the search space of feasible policies; for another, it might be safer to make O_i^- only a subset O_i, allowing some space for the agents to coordinate among themselves.

Lastly, as a supplement, we provide some elaboration on when and why no feasible policy exists, drawing inspiration from the experimental results in the *myopic preference* scenario.

Definition 2 (Crossroads and Streets)**.** *In a 2D grid map, given a pair of goals $g_1 = (x_1, y_1)$ and $g_2 = (x_2, y_2)$, we call the locations (x_1, y_2) and (x_2, y_1) crossroads. If the crossroads are not on the boundary of the given map layout, we call them interior crossroads. And also, g_1 and g_2 are said to be on the same street if $x_1 = x_2 \vee y_1 = y_2$.*

Theorem 2. *Given a map without obstacles, a feasible policy profile exists for two agents with sensors of range two, if their goals forms no interior crossroads and are not on the same street.*

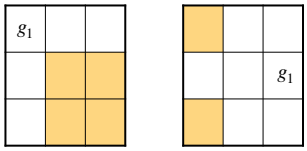


Figure 3: Orange cells denote the choices of g_2 that lead to feasible policies.

We note that the above theorem preliminarily characterizes a *sufficient* condition for the existence of feasible policies in two-agent cases and provides an illustrative example in Figure 3. The formal proof is deferred to Appendix F, while we provide an application as follows. By applying the above theorem to a 6×6 map with two agents, one can use simple combinatorics to calculate that there are $4 \times 4 \times 4 \times 2 + 5 \times 6 \times 2 \times 2 - 4 = 244$ goal profiles that satisfy the condition, which matches the actual experimental results for the *myopic preference* scenario in Table 4 (similarly for 192 for 5×6 configurations and 300 for 6×7 configurations). As the *myopic preference* heuristic imposes additional constraints on the original problem, then this feasibility result must hold in general without these constraints.

5.2.2 Traffic Rules

In addition to those single-agent heuristics developed on O_i^-, we also intend to indentify some common behavioral patterns on O_i^+ , inspired by real-world traffic systems. When a driver is approaching traffic lights, her driving decisions are made according to certain traffic rules based on her relative position to the other cars. In this context, a set of traffic rules serves as a common protocol that all drivers must obey and is viewed from a relative frame. In real-world situations, designers first come up with certain rules and then make all drivers aware of them. In this section, we investigate this idea in a reversed manner by asking the question: *Ensuring the whole system to work, is there any underlying traffic rule that can be found?* Let $f : \bigcup_i O_i^+ \mapsto O^+$ denote a function that transforms the local states where agents detect each other into those viewed from a unified relative frame. A *traffic rule* $tr : O^+ \mapsto A$ then maps these transformed local states to available actions, and is adopted by all agents as common knowledge. Once the complementary policy $\pi_i^- : O_i^- \mapsto A$ is determined, the final policy profile can be collectively formed as $\{\pi_i\}_{i\in N} = \{\pi_i^- \cup (tr \circ f)\}_{i\in N}$. We explore two alternatives for encoding these traffic rules, with detailed encoding deferred to Appendix D.

1. The traffic rule depends on both the location of the agent and its relative observations of others. One can opt to impose the *default action* heuristic on those local states where agents do not detect any others. In this case, all agents are actually adopting the same way of routing (subject to different goals) when they have not encountered each other, and then follow the same protocol when they do meet. As shown in Table 5, even when *default actions* are preferred, certain traffic rules can still be identified with the help of sensors with a range of only two.
2. The traffic rule depends solely on the agent's relative observations of others, regardless of its own location. Surprisingly, completely opposite results are revealed in Table 5. No such traffic rule is identified, even when we do not require agents to follow the *default action* heuristic and equip them with the most powerful sensors (i.e., sensor 5, which can cover the entire 6×6 map). One reasonable explanation is that, unlike real-world traffic systems, our artificial map is bounded, and thus, agents are expected to behave differently as they approach the boundary, rather than behaving the same way as they do in the interior of the map.

Scenario (map, agents)	#found/#total
Sensor 2, 3, 4 or 5	
#1 $(6\times 6, 2)$	1260/1260
#1 $(6\times 6, 2)$+default	1260/1260
#2 $(6\times 6, 2)$	0/1260
#2 $(6\times 6, 2)$+default	0/1260

Table 5: Certain traffic rules exist when they depend on agents' locations, but no such traffic rules can be found if they are independent of agents' locations. For both alternatives, For both alternatives, we verify the existence of both the vanilla version and the *default action* version.

5.3 Anytime Optimization

When it comes to policy optimization, the system designer can also specify human-understandable rules for π_i^- in the first place, and the optimization procedures are therefore only responsible for (optimally) "completing" the rest π_i^+. Recall that we have designed several action heuristics for π_i^- in Section 5.2.

Map (#agents)	from scratch			default action			last-minute coordination			
	start	end	↓	start	end	↓	start	end	↓	$\downarrow^{total}$
Sensor 2										
3×3 (2)	360	181	49.72%	415	263	36.63%	264	203	23.11%	43.61%
4×4 (2)	4298	2732	36.44%	2653	2473	6.78%	1250	1241	0.72%	71.13%
5×5 (2)	15248	15149	0.65%	7349	7302	0.64%	4436	4422	0.32%	71.00%
6×6 (2)	43355	41007	5.42%	17734	17331	2.27%	11420	11420	0.00%	73.66%
Sensor 3										
4×4 (2)	3391	3379	0.35%	3368	3256	3.33%	1462	1441	1.44%	57.51%
5×5 (2)	14962	14746	1.44%	10423	10105	3.05%	4605	4539	11.43%	69.66%
6×6 (2)	52530	45953	12.52%	26873	26873	0.00%	11544	11526	0.16%	78.06%
Sensor 4										
5×5 (2)	22801	21950	3.73%	14359	11277	21.46%	4894	4882	0.25%	78.59%
6×6 (2)	54669	48316	11.62%	29242	29238	0.01%	11669	11628	0.35%	78.73%

Table 6: Optimization results starting from different action heuristics. The subtitle "start" means the sum-of-makespan of first feasible policy while "end" means that of the last one, after which no better one has been found for 2 hours. "Default action" and "last-minute coordination" correspond to the respective heuristics in Section 5.2.

As reported in Table 6, we experiment on optimizing the policy from scratch as well as with different action heuristics . The program terminates after a sufficiently long period without finding a better solution. This indicates that the improvement achieved through optimization is significantly less than that obtained by incorporating effective heuristics. A lesson taken from this table is that, to compute better policies, attention should be paid to identifying helpful constraints or properties compatible with the (near-)optimal policies, rather than solely relying on the solver for the entire optimization process.

6 Conclusion

This paper presents the implementation of an ASP-based system for computing universal plans for any arbitrary 2D map and a group of partially observable agents with specific goals. *We emphasize again that our solution concept, which consists of a collection of decentralized policies, can suggest immediate actions whenever agents encounter any contingency. This is the key contribution that distinguishes our work from the existing literature, particularly in the area of MAPF.* The system is used for comprehensive studies on the behavioral patterns of the agents. With the help of ASP, user-specified preferences can be easily represented, thus improving the policy. Furthermore, we highlight several potential directions and applications for our system.

- Despite being able to solve larger problem instances compared to existing Dec-POMDP solvers, real-world applications may require even larger scales that are currently intractable for our system. One possible approach is to leverage our system to compute policies for (local) multi-agent coordination in sub-regions, and utilize other single-agent planners to guide agents while they are operating alone.
- Another valuable application is to integrate our rule-based system into a learning framework. Some work has explored a similar idea by manually designing *acceleration rules* and *safety rules* for single-agent RL tasks [8]. Additionally, our system provides insights into how to automate the generation of rules to support learning in multi-agent domains.

Acknowledgments

We thank Jiuzhi Yu for his invaluable efforts and unwavering support in the initial stages of this work. We also thank Yuxin Pan and the anonymous reviewers for their insightful comments.

References

[1] Christopher Amato, Daniel S Bernstein & Shlomo Zilberstein (2010): *Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs*. Autonomous Agents and Multi-Agent Systems 21, pp. 293–320. Available at https://doi.org/10.1007/s10458-009-9103-z.

[2] Daniel S Bernstein, Christopher Amato, Eric A Hansen & Shlomo Zilberstein (2009): *Policy iteration for decentralized control of Markov decision processes*. Journal of Artificial Intelligence Research 34, pp. 89–132. Available at https://dl.acm.org/doi/abs/10.5555/1622716.1622720.

[3] Aysu Bogatarkan & Esra Erdem (2020): *Explanation generation for multi-modal multi-agent path finding with optimal resource utilization using answer set programming*. Theory and Practice of Logic Programming 20(6), pp. 974–989. Available at https://doi.org/10.1017/S1471068420000320.

[4] Aysu Bogatarkan, Volkan Patoglu & Esra Erdem (2019): *A Declarative Method for Dynamic Multi-Agent Path Finding*. In: GCAI, pp. 54–67. Available at https://doi.org/10.29007/cnzw.

[5] Blai Bonet & Hector Geffner (2011): *Planning under partial observability by classical replanning: Theory and experiments*. In: Twenty-Second International Joint Conference on Artificial Intelligence. Available at https://dl.acm.org/doi/10.5555/2283696.2283725.

[6] Esra Erdem, Doga Kisa, Umut Oztok & Peter Schüller (2013): *A general formal framework for pathfinding problems with multiple agents*. In: Proceedings of the AAAI Conference on Artificial Intelligence, 27, pp. 290–296. Available at https://doi.org/10.1609/aaai.v27i1.8592.

[7] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli & Shimon Whiteson (2018): *Counterfactual multi-agent policy gradients*. In: Proceedings of the AAAI conference on artificial intelligence, 32. Available at https://dl.acm.org/doi/10.5555/3504035.3504398.

[8] Zihang Gao, Fangzhen Lin, Yi Zhou, Hao Zhang, Kaishun Wu & Haodi Zhang (2020): *Embedding high-level knowledge into dqns to learn faster and more safely*. In: Proceedings of the AAAI Conference on Artificial Intelligence, 34, pp. 13608–13609. Available at https://doi.org/10.1609/aaai.v34i09.7091.

[9] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2019): *Multi-shot ASP solving with clingo*. Theory and Practice of Logic Programming 19(1), pp. 27–82. Available at https://doi.org/10.1017/S1471068418000054.

[10] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub & Marius Schneider (2011): *Potassco: The Potsdam answer set solving collection*. Ai Communications 24(2), pp. 107–124. Available at https://dl.acm.org/doi/abs/10.5555/1971622.1971623.

[11] Martin Gebser, Benjamin Kaufmann, André Neumann & Torsten Schaub (2007): *clasp: A conflict-driven answer set solver*. In: Logic Programming and Nonmonotonic Reasoning: 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007. Proceedings 9, Springer, pp. 260–265. Available at https://doi.org/10.1007/978-3-540-72200-7_23.

[12] Michael Gelfond & Vladimir Lifschitz (1988): *The stable model semantics for logic programming.* In: ICLP/SLP, 88, Cambridge, MA, pp. 1070–1080.

[13] Rodrigo N Gómez, Carlos Hernández & Jorge A Baier (2020): *Solving sum-of-costs multi-agent pathfinding with answer-set programming*. In: Proceedings of the AAAI Conference on Artificial Intelligence, 34, pp. 9867–9874. Available at https://doi.org/10.1609/aaai.v34i06.6540.

[14] Eric A Hansen, Daniel S Bernstein & Shlomo Zilberstein (2004): *Dynamic programming for partially observable stochastic games*. In: *AAAI*, 4, pp. 709–715. Available at https://dl.acm.org/doi/10.5555/1597148.1597262.

[15] Leslie Pack Kaelbling, Michael L Littman & Anthony R Cassandra (1998): *Planning and acting in partially observable stochastic domains*. *Artificial intelligence* 101(1-2), pp. 99–134. Available at https://doi.org/10.1016/S0004-3702(98)00023-X.

[16] Wietze Koops, Sebastian Junges & Nils Jansen (2024): *Approximate dec-POMDP solving using multi-agent A**. In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 6743–6751. Available at https://dl.acm.org/doi/10.24963/ijcai.2024/745.

[17] Akshat Kumar, Hala Mostafa & Shlomo Zilberstein (2016): *Dual formulations for optimizing Dec-POMDP controllers*. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, 26, pp. 202–210. Available at https://doi.org/10.1609/icaps.v26i1.13759.

[18] Jiaoyang Li, Wheeler Ruml & Sven Koenig (2021): *Eecbs: A bounded-suboptimal search for multi-agent path finding*. In: *Proceedings of the AAAI conference on artificial intelligence*, 35, pp. 12353–12362. Available at https://doi.org/10.1609/aaai.v35i14.17466.

[19] Michael L Littman (2009): *A tutorial on partially observable Markov decision processes*. *Journal of Mathematical Psychology* 53(3), pp. 119–125. Available at https://doi.org/10.1016/j.jmp.2009.01.005.

[20] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel & Igor Mordatch (2017): *Multi-agent actor-critic for mixed cooperative-competitive environments*. *Advances in neural information processing systems* 30. Available at https://dl.acm.org/doi/10.5555/3295222.3295385.

[21] Omid Madani, Steve Hanks & Anne Condon (1999): *On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems*. In: *AAAI/IAAI*, Citeseer, pp. 541–548. Available at https://dl.acm.org/doi/10.5555/315149.315395.

[22] Victor W Marek & Miroslaw Truszczyński (1999): *Stable models and an alternative logic programming paradigm*. In: *The Logic Programming Paradigm: a 25-Year Perspective*, Springer, pp. 375–398. Available at https://doi.org/10.1007/978-3-642-60085-2_17.

[23] Ilkka Niemelä (1999): *Logic programs with stable model semantics as a constraint programming paradigm*. *Annals of mathematics and Artificial Intelligence* 25, pp. 241–273. Available at https://doi.org/10.1023/A:1018930122475.

[24] Keisuke Okumura (2023): *Lacam: Search-based algorithm for quick multi-agent pathfinding*. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 37, pp. 11655–11662. Available at https://doi.org/10.1609/aaai.v37i10.26377.

[25] Keisuke Okumura, Manao Machida, Xavier Défago & Yasumasa Tamura (2022): *Priority inheritance with backtracking for iterative multi-agent path finding*. *Artificial Intelligence* 310, p. 103752. Available at https://doi.org/10.1016/j.artint.2022.103752.

[26] Frans A Oliehoek & Christopher Amato (2016): *A concise introduction to decentralized POMDPs*. Springer. Available at https://doi.org/10.1007/978-3-319-28929-8.

[27] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster & Shimon Whiteson (2020): *Monotonic value function factorisation for deep multi-agent reinforcement learning*. *Journal of Machine Learning Research* 21(178), pp. 1–51. Available at https://dl.acm.org/doi/abs/10.5555/3455716.3455894.

[28] Rasmus Rendsvig & John Symons (2021): *Epistemic Logic*. In Edward N. Zalta, editor: *The Stanford Encyclopedia of Philosophy*, Summer 2021 edition, Metaphysics Research Lab, Stanford University.

[29] Marcel Schoppers (1987): *Universal Plans for Reactive Robots in Unpredictable Environments*. In: *IJCAI*, 87, Citeseer, pp. 1039–1046. Available at https://dl.acm.org/doi/10.5555/1625995.1626091.

[30] Guni Sharon, Roni Stern, Ariel Felner & Nathan R Sturtevant (2015): *Conflict-based search for optimal multi-agent pathfinding*. Artificial Intelligence 219, pp. 40–66. Available at https://doi.org/10.1016/j.artint.2014.11.006.

[31] Tran Cao Son, Enrico Pontelli, Marcello Balduccini & Torsten Schaub (2023): *Answer Set Planning: A Survey*. Theory and Practice of Logic Programming 23(1), pp. 226–298. Available at https://doi.org/10.1017/S1471068422000072.

[32] Roni Stern (2019): *Multi-agent path finding–an overview*. Artificial Intelligence, pp. 96–115. Available at https://doi.org/10.1007/978-3-030-33274-7_6.

[33] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar et al. (2019): *Multi-agent pathfinding: Definitions, variants, and benchmarks*. In: Twelfth Annual Symposium on Combinatorial Search. Available at https://doi.org/10.1609/socs.v10i1.18510.

[34] Pavel Surynek (2019): *Unifying search-based and compilation-based approaches to multi-agent path finding through satisfiability modulo theories*. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, pp. 1177–1183. Available at https://doi.org/10.24963/ijcai.2019/164.

[35] Pavel Surynek, Ariel Felner, Roni Stern & Eli Boyarski (2016): *Efficient SAT approach to multi-agent path finding under the sum of costs objective*. In: Proceedings of the twenty-second european conference on artificial intelligence, pp. 810–818. Available at https://doi.org/10.3233/978-1-61499-672-9-810.

[36] Lifschitz Vladimir (2008): *What is answer set programming*. In: Proc. Natl. Conf. Artif. Intell, 3, pp. 1594–1597.

[37] Jiangxing Wang, Jiaoyang Li, Hang Ma, Sven Koenig & S Kumar (2019): *A new constraint satisfaction perspective on multi-agent path finding: Preliminary results*. In: 18th International Conference on Autonomous Agents and Multi-Agent Systems. Available at https://dl.acm.org/doi/10.5555/3306127.3332075.

[38] Ibrahim Faruk Yalciner, Ahmed Nouman, Volkan Patoglu & Esra Erdem (2017): *Hybrid conditional planning using answer set programming*. Theory and Practice of Logic Programming 17(5-6), pp. 1027–1047. Available at https://doi.org/10.1017/S1471068417000321.

[39] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen & Yi Wu (2022): *The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games*. In: Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track. Available at https://dl.acm.org/doi/abs/10.5555/3600270.3602057.

[40] Jingjin Yu & Steven M LaValle (2013): *Structure and intractability of optimal multi-robot path planning on graphs*. In: Twenty-Seventh AAAI Conference on Artificial Intelligence. Available at https://doi.org/10.1609/aaai.v27i1.8541.

[41] Bruno Zanuttini, Jérôme Lang, Abdallah Saffidine & François Schwarzentruber (2020): *Knowledge-based programs as succinct policies for partially observable domains*. Artificial Intelligence 288, p. 103365. Available at https://doi.org/10.1016/j.artint.2020.103365.

A ASP Encoding for the Problem

In this section, we show how to encode the problem by ASP. For the simplicity of presentation, we present `p:- q` in the meaning of $p \leftarrow q$. First, we encode the following facts:

- `cell(X,Y)` and `block(X,Y)` encode the empty cells and obstacles on the map, respectively.
- `goali((GX,GY))` specifies the goal designated to agent i.

Actions are encoded as follows (similarly for `down`, `right`, `left` and `nil`).

```
action(X,Y,up):- cell(X,Y), not block(X,Y),
    cell(X-1,Y), not block(X-1,Y).
```

Then we encode the global state representation, which is a snapshot of the whole system including the current positions and the respective goals of all agents. Note that all agents are supposed to be placed at an empty cell and not to overlap with each other.

```
gState(S):- S=(L1,...,Ln,G1,...,Gn), goal1(G1), ..., goaln(Gn),
    Li=(Xi,Yi), cell(Xi,Yi), not block(Xi,Yi), % for all i
    Li!=Lj. % for all distinct i,j -> no vertex conflict
```

Upon the definition of global states, we then define the transition. A global transition is triggered upon a collection of all agents' reported actions. Note that any pair of agents cannot traverse the same passage, i.e. no immediate swap is allowed.

```
trans(S1,A1,...,An,S2):- gState(S1), gState(S2),
    S1=(L11,...,L1n,G11,...,G1n), S2=(L21,...,L2n,G21,...,G2n),
    L11=(X11,Y11), ..., L1n=(X1n,Y1n),
    L21=(X21,Y21), ..., L2n=(X2n,Y2n),
    move(X1i,Y1i,Ai,X2i,Y2i), % for all i
    (L1i,L1j)!=(L2j,L2i). % for all distinct i,j -> no edge conflict
```

To make the system goal-directed, we also need to define the global goal state.

```
goal_gState(S):- gState(S), S=(G1,...,Gn,G1,...,Gn),
    goal1(G1), ..., goaln(Gn).
```

As we mentioned earlier, each agent is only capable of observing others within her local FoV.

```
aState(AS):- AS=(Self,O2,...,On,Goal),
    Self=(X,Y), cell(X,Y), not block(X,Y),
    Goal=(Xg,Yg), cell(Xg,Yg), not block(Xg,Yg),
    near(Self,Oi). % for all i=2..n
```

The `near/2` predicate denotes the functionality of the sensor. If the opponent is beyond the detection of the agent, we let the constant `empty` be the placeholder for such a null position. The constant `R` denotes the sensor range. One can alternatively adopt any other distance measure, e.g. Manhattan or Euclidean distance. For simplicity, our example adopts the L_∞ metric as the distance measure, i.e. $dist(p,q) = \max\{|p_x - q_x|, |p_y - q_y|\}$.

```
near(Self,empty):- Self=(X1,Y1), cell(X1,Y1).
near(Self,Other):- Self!=Other, Self=(X1,Y1), Other=(X2,Y2),
    cell(X1,Y1), not block(X1,Y1), cell(X2,Y2), not block(X2,Y2),
    |X1-X2| <= R, |Y1-Y2| <= R.
```

As required, each agent will not be incentivized to do any further move once she reaches the goal. Thus, each agent should be capable of recognizing her own dedicated goal.

```
% for all i
goali_aState(AS):- aState(AS), AS=(Gi,_,...,_,Gi), goali(Gi).
```

To map from global states to each agent's local states, we then define the observation model. Note that by `(L1,...,Ln)\Self` we mean to exclude the agent herself from the vector of all agents' locations. By `empty/Oi` we mean that the i-th one of the other agents can be outside or inside the FoV. Correspondingly, if `Oi` is not detected, then we include `not near(Self,Oi)`, otherwise no leading `not`. The observation model is in fact the most expensive one to encode in our system, since we need to enumerate all possible 2^{n-1} situations of sensor detection for each agent, thus, $n \cdot 2^{n-1}$ axioms in total.

```
obsi(S,AS):- gState(S), S=(L1,...,Ln,G1,...,Gn),
    Self=Li, (O2,...,On)=(L1,...,Ln)\Self,
    aState(AS), AS=(Self,empty/O2,...,empty/On,Gi),
    {not} near(Self,Oi). % for all i=2..n
```

On top of the above, we formulate each individual policy under two restrictions: 1) once an agent reaches her dedicated goal, she has to stop; 2) pick one single available action otherwise. Our desired solution is consequently a deterministic one and executable from the local perspective of an agent.

```
doi(AS,nil):- goali_aState(AS).
{doi(AS,A): avai_action(AS,A)}=1 :- aState(AS),
    AS=(Self,_,...,_,Goal),
    Self != Goal, goali(Goal).
```

To ensure that the policy works for every contingency, we inductively define the reachability of each global state given the policy. Basically, a goal global state is trivially reachable to itself. Inductively, any global state is reachable to the goal state if it can, according to the policy, transit to a successor state that is reachable to goal state.

```
reached(S):- goal_gState(S).
reached(S1):- gState(S1), trans(S1,A1,...,An,S2), reached(S2),
    obsi(Si,ASi), doi(ASi,Ai). % for all i
```

Finally, we include the integrity constraint to rule out infeasible policies.

```
:- gState(S), not reached(S).
```

B Existence of Policies for Some Manually Designed Maps

To investigate how the existence of feasible policies may depend on the capability of sensors, we conduct experiments on six manually designed map layouts, as shown in Table 7. We evaluate the feasibility over

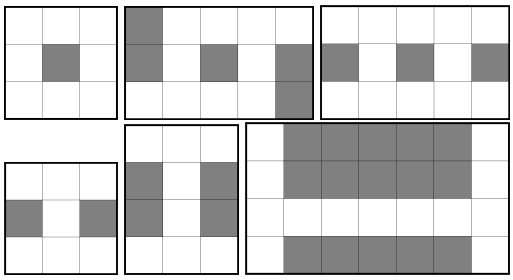


Figure 4: Manually designed layouts for the results reported in Table 7. From the left to the right and the top to the bottom, layouts are identified as `id_1` to `id_6`.

Map (#agents)	sensor 1 #feasible/#proper/#total	sensor 2 #feasible/#proper/#total	full sensor #feasible/#proper/#total
`id_1`(2)	28/56/56	56/56/56	no need
`id_2`(2)	46/56/90	56/56/90	56/56/90
`id_3`(3)	0/168/1320	168/168/1320	168/168/1320
`id_4`(3)	0/24/210	24/24/210	no need
`id_5`(3)	0/24/336	24/24/336	24/24/336
`id_6`(3)	0/24/1716	24/24/1716	24/24/1716

Table 7: Feasibilities with different sensors, for these manually designed layouts in Figure 4.

all possible *proper* goal profiles for agents with sensors of different ranges. Detailed map layouts are shown in Figure 4. Map `id_1` and `id_2` are designed with roundabouts for two-agent cases, and map `id_3` – `id_6` are designed to further include channels for three agent cases. Notice that except for map `id_1` and `id_4`, agents need sensors of larger range to cover the whole map but actually sensors of range two already suffice. *Full sensors* are the ones that can cover the whole map.

C ASP Encoding for Action Preferences

C.1 The `cost` Predicate

Let the predicate `cost(AS,A,C)` define the Manhattan distance `C` from the local state `AS` to the goal by taking action `A`. The Manhattan distance is computed from the perspective of the agent herself, without any prescription of the actions that will be taken by other agents.

Below shows an example how we encode Manhattan distance for two agents (encodings for $n > 2$ agents can be easily generalized). If the action will not lead to any immediate collision, the cost will simply be the Manhattan distance towards the goal.

```
cost(AS,Action,C):-
    aState(AS), AS=(Self,empty,Goal),
    Self=(X,Y), Goal=(Xg,Yg),
    move(X,Y,Action,Xs,Ys),
    C=1+|Xg-Xs|+|Yg-Ys|.

```

```
cost(AS,Action,C):-
    aState(AS), AS=(Self,Other,Goal),
    near(Self,Other),
    not nextTo(Self,Action,Other),
    Self=(X,Y), Goal=(Xg,Yg),
    move(X,Y,Action,Xs,Ys),
    C=1+|Xg-Xs|+|Yg-Ys|.

cost(AS,Action,INFTY):-
    aState(AS), AS=(Self,Other,Goal),
    near(Self,Other),
    nextTo(Self,Action,Other),
    Self=(X,Y), Others2=(Xs,Ys),
    move(X,Y,Action,Xs,Ys).

nextTo(Loc1, Action, Loc2) :- Loc1 = (X1, Y1), Loc2 = (X2, Y2),
    cell(X1, Y1),notblock(X1, Y1),
    cell(X2, Y2),notblock(X2, Y2),
    move(X1, Y1, Action, X2, Y2).
```

Otherwise, agents may run into walls or go beyond the map, also incurring a cost of infinity.

```
 cost(AS,Action,INFTY):-
     aState(AS), isAction(Action),
     not avai_action(AS,Action).
```

Note that this cost metric is defined in such a fashion that an agent prefers actions with the least Manhattan distance without regard to the other agents' true subsequent actions.

C.2 Three Types of Preferences

For *default actions*, we modify the policy definition[6] as

```
{doi(AS,A):avai_action(AS,A)}=1 :-
    aState(AS), AS=(Self,Other,Goal),
    near(Self,Other), Other!=empty,
    Self!=Goal, goali(Goal).

{doi(AS,A): avai_action(AS,A),
    cost(AS,A,C0),
    cost(AS,up,C1), ..., cost(AS,nil,C5),
    C0<=C1,...,C0<=C5}=1 :-
    aState(AS), AS=(Self,empty,Goal),
    Self!=Goal, goali(Goal).
```

For *last-minute coordination*, we further sophisticate the policy definition as

[6] without particular specification, we usually keep the stop-at-goal rule unchanged.

```
{doi(AS,A):avai_action(AS,A)}=1 :-
    aState(AS), AS=(Self,Other,Goal),
    near(Self,Other), Other!=empty,
    lastmin(Self,Other),
    Self!=Goal, goal2(Goal).

{doi(AS,A):avai_action(AS,A),
    cost(AS,A,C0),
    cost(AS,up,C1), ..., cost(AS,nil,C5),
    C0<=C1, ..., C0<=C5}=1 :-
    aState(AS), AS=(Self,Other,Goal),
    near(Self,Other), Other!=empty,
    not lastmin(Self,Other),
    Self!=Goal, goali(Goal).

{doi(AS,A):avai_action(AS,A),
    cost(AS,A,C0),
    cost(AS,up,C1), ..., cost(AS,nil,C5),
    C0<=C1, ..., C0<=C5}=1 :-
    aState(AS), AS=(Self,empty,Goal),
    Self!=Goal, goali(Goal).
```

where the `lastmin` predicate defines the moments when two agents are about to collide with each other if both make inappropriate actions, i.e., they are two units of Manhattan distance away from each other.

```
lastmin(P1,P2):-
    P1=(X1,Y1), P2=(X2,Y2),
    cell(X1,Y1), cell(X2,Y2),
    |X1-X2|+|Y1-Y2|<=2.
```

For *myopic preferences*, we do not have to discuss that many sub-cases, therefore, can simply write

```
{doi(AS,A):avai_action(AS,A),
    cost(AS,A,C0),
    cost(AS,up,C1), ..., cost(AS,nil,C5),
    C0<=C1, ..., C0<=C5}=1 :-
    aState(AS), AS=(Self,_,Goal),
    Self!=Goal, goali(Goal).
```

D ASP Encoding for Traffic Rules

For the first alternative, the policy definitions for π_i^+ are modified as follows. Note that we omit the definitions for π_i^- here. One can opt to impose the *default action* heuristic on those local states where agents have not detected any of the others.

```
{traffic(Self,Rel_pos,A): avai_action(AS,A)}=1 :-
```

```
    aState(AS), AS=(Self,Other,Goal),
    near(Self,Other), Other!=empty,
    Self=(X1,Y1), Other=(X2,Y2),
    Rel_pos=(X2-X1,Y2-Y1),
    Self != Goal.

{doi(AS,A): traffic(Self,Rel_pos,A)}=1 :-
    aState(AS), AS=(Self,Other,Goal),
    near(Self,Other), Other!=empty,
    Self=(X1,Y1), Other=(X2,Y2),
    Rel_pos=(X2-X1,Y2-Y1),
    Self!=Goal, goali(Goal).
```

For the second alternative, we instead include the following rules. And also, one can implement π_i^- as the *default action* version.

```
{traffic(Rel_pos,A): avai_action(AS,A)}=1 :-
    aState(AS), AS=(Self,Other,Goal),
    near(Self,Other), Other!=empty,
    Self=(X1,Y1), Other=(X2,Y2),
    Rel_pos=(X2-X1,Y2-Y1),
    Self!=Goal.

{doi(AS,A): traffic(Rel_pos,A)}=1 :-
    aState(AS), AS=(Self,Other,Goal),
    near(Self,Other), Other!=empty,
    Self=(X1,Y1), Other=(X2,Y2),
    Rel_pos=(X2-X1,Y2-Y1),
    Self!=Goal, goali(Goal).
```

E Elaboration on Table 4

One can easily run experiments to check if there always exist any feasible universal policies in small domains, e.g. two agents in 6-by-6 grid worlds with no obstacles. Our experiments on 6-by-6, 5-by-6 and 6-by-7 empty grid maps with two agents have shown that feasible policies exist for goal profiles that are in one of the following conditions:

1. If one of the goal is at the corner, the other goal can be at any cell within the submap induced by eliminating the column and row where the former goal is located. As shown in Figure 5(a), suppose g_1 is at the upper left corner, then g_2 can be any one of cells in the orange space.
2. If one of the goal is at one of the four boundaries other than the corner, the other goal can only be at any cell that are located at the opposite boundary other than the cell that are in the same line. As shown in Figure 5(b), suppose g_1 is at the rightmost boundary, then g_2 can be any one of cells in the orange space. Same situations happen when g_1 is at the one of other three boundaries.

Take the 6-by-6 map with two agents as an example, we have totally $(6\times 6)\times(6\times 6-1)=1260$ proper goal profiles. According to the description above, we have $4\times 4\times 4\times 2+5\times 6\times 2\times 2-4=244$

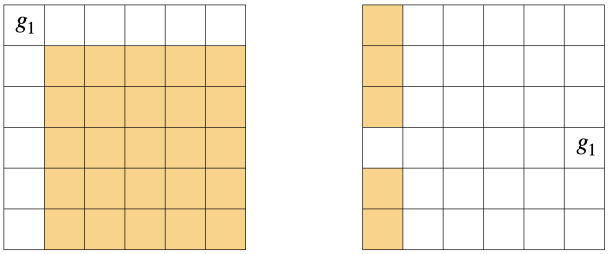


Figure 5: Orange cells denote possible g_2's that lead to feasible policies. (a) Left: the corner case; (b) Right: the boundary case (other than corners).

goal profiles that allow feasible policies to exist, which matches the actual experimental results (192 for 5-by-6 layouts, 244 for 6-by-6 layouts, and 300 for 6-by-7 layouts).

F Elaboration on Theorem 2

Further, based on experimental findings, we will then formally characterize the results for maps of any size with two agents, starting with a few definitions.

Definition 3. *(Locks). If a group of agents repeatedly go through the same set of global states, they are said to be in locks. This particular set of global states are called a lock set. If the lock set contains more than one states, agents are said to be in a live lock. Otherwise, they are said to be in a dead lock.*

For example, Figure 6(a) has shown a case of a live lock. Agents start from red ones, simultaneously go to green ones and get back to red ones. The lock set contains two states $\{[(4,B),(4,D)],[(3,B),(5,D)]\}$. An example of deadlock is also shown in Figure 6(b) if two agents simultaneously decide to stop, then they will consequently both stop forever since their policies only depend the current local states, leading to a lock set of $\{[(3,C),(4,D)]\}$.

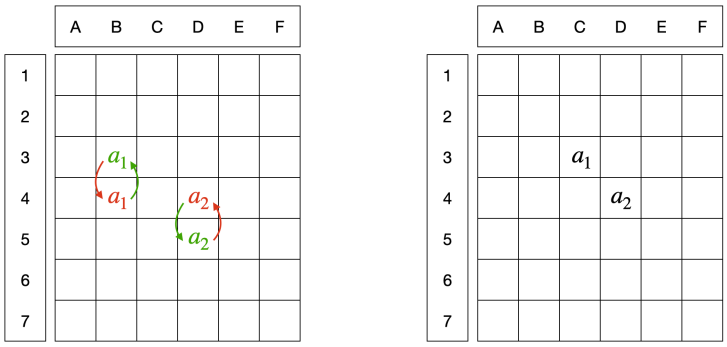


Figure 6: (a) Left: an example of a live lock; (b) Right: an example of a dead lock if both stop.

Proposition 2. *The rule that encodes myopic preferences will rule out any live lock.*

The proposition is way clear to prove. For a live lock to exist, there should be at least two global states in the lock set, which means at least one of the agents would visit her previous local state (different from the current one). Nevertheless, according to the definition of *myopic preferences*, even if an agent's most preferred action is blocked by another agent, she would stop rather than going backwards, which results in a less Manhattan distance.

Recall that in Definition 2, we define *crossroads* and *streets*. To illustrate with an example, Figure 7(a) has shown a typical case where cells in orange are the columns/rows where goals are located, and the orange cells are the consequent interior crossroads. In Figure 7(b), the two goals are in the same street marked in green as well.

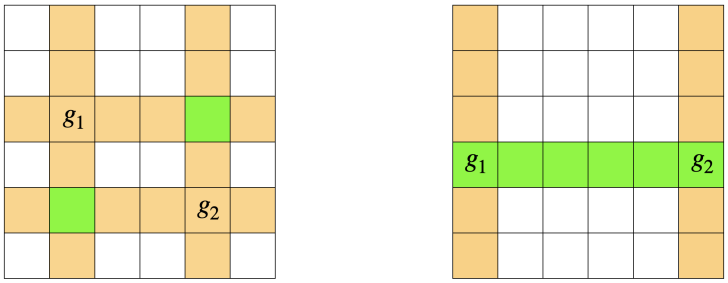


Figure 7: Green cells denote crossroads. (a) Left: a typical case of crossroads; (b) Right: a degraded case when two goals are in the same column/row.

We then elaborate why *crossroads* and *streets* are crucial for feasible policies. For example, Figure 8(a) has shown that those blue agents a_1 and a_2 are running into an interior crossroad, and at the next step they will collide into the each other at this crossroad according to the rule of *myopic preference*. Note that for red ones in Figure 8(a) can avoid collisions by both going down since it is indeed one of the most preferred actions, but then they will be faced with the same situation as the blue ones. Similar situation happens in Figure 8(b) for the blue a_1 and the red a_2 as they will collide into each other at the next step, while the blue a_1 and blue a_2 will simply run into a dead lock as they will both choose to temporally stop but will turn out to stop forever. For Figure 8(c) and (d), these goals do not form any interior crossroad and are not on the same street, and by experiments such configurations are solvable, which we will also prove.

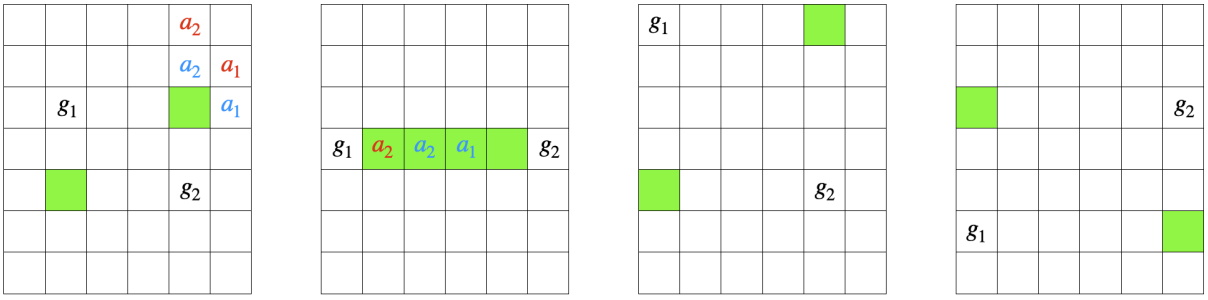


Figure 8: (a) a typical case causing collisions; (b) a degraded case causing dead locks; (c) a corner case; (d) a boundary case

We first prove a proposition about dead locks.

Proposition 3. *Given a map without obstacles with two agents, if a goal profile forms no interior crossroads, dead locks will only be formed iff two goals are on the same street, under the myopic preferences.*

Proof.
($\Leftarrow$) If two goals are on the same street, any two adjacent locations in between will cause a dead lock, given the *myopic preference*.

($\Rightarrow$) We prove the contraposition. If two goals are not on the same street, there will not exist such a case when both agents have to stop, i.e., at least one agent has a better action moving itself closer to its goal. □

We can now prove Theorem 2 as follows. It can be shown that such policy exists even with *myopic preferences*, then it must also hold in general without this constraints.

Proof. (of Theorem 2) For any of the two agents,

1. If the only non-stop action that is the most preferred under the *myopic preference* is blocked by the other adjacent agent, then it will stop to wait for exactly one time step, since there will no dead locks according to Proposition 3.
2. If it is not the case of 1., there are at least two tied actions that are both the most preferred under the *myopic preference*. Then there must be one of these two actions that will not cause any collision at the next step.
3. Based on 1., 2. and Proposition 2, we can see that for every time step, there is at least one agent shortening her distance towards her goal. Given that the map is finite, both agents will reach their goals in finite time steps.

□

Moreover, we can claim that there is nothing to do with the shape of the map for Theorem 2 to hold.

The Combined Problem of Online Task Assignment and Lifelong Path Finding in Logistics Warehouses: Rule-Based Systems Matter

Fengming Zhu[1] Weijia Xu[2] Yifei Guo[2] Fangzhen Lin[1]
[1]CSE Department, Hong Kong University of Science and Technology, Hong Kong SAR, China
[2]Meituan Academy of Robotics Shenzhen, Shenzhen, China
fzhuae@connect.ust.hk {xuweijia, guoyifei02}@meituan.com flin@cs.ust.hk

We study the combined problem of online task assignment and lifelong path finding, which is crucial for the logistics industries. However, most literature either (1) focuses on lifelong path finding assuming a given task assigner, or (2) studies the offline version of this problem where tasks are known in advance. We argue that, to maximize the system throughput, the online version that integrates these two components should be tackled directly. To this end, we introduce a formal framework of the combined problem and its solution concept. Then, we design a rule-based lifelong planner under a practical robot model that works well even in environments with severe local congestion. Upon that, we automate the search for the task assigner with respect to the underlying path planner. Simulation experiments conducted in warehouse scenarios at *Meituan*, one of the largest shopping platforms in China, demonstrate that (a) *in terms of time efficiency*, our system requires only 83.77% of the execution time needed for the currently deployed system at Meituan, outperforming other SOTA algorithms by 8.09%; (b) *in terms of economic efficiency*, ours can achieve the same throughput with only 60% of the agents currently in use. The code and demos are available at `https://github.com/Fernadoo/Online-TAPF`.

1 Introduction

We consider the problem present in highly automated real-world warehouses where a fleet of robots is programmed to pick up and deliver packages without any collision. This is a significant problem for logistics companies as it has a major impact on their operational efficiency. It is a difficult problem for at least the following two reasons: (1) the computational complexity of multi-agent path finding is notoriously high, especially when the number of robots is large, and (2) the dynamic and real-time assignment of tasks to the robots both depends on and affects the subsequent path finding.

There is a vast literature that studies idealized abstractions of such real-world problems. The most commonly seen formulation is to assume a given (or naive) task assigner, and therefore, the focus is merely on the path-finding part, which is usually termed as one-shot *Multi-Agent Path Finding* (MAPF) [35, 5, 26, 13, 22]or its lifelong version *Multi-Agent Pickup and Delivery* (MAPD) [18, 30, 14, 22]. However, to maximize the throughput of the whole production pipeline, the task assigner should also be deliberately designed with respect to the particular underlying path planner. To this end, some recent work has further investigated the combined problem of *Task Assignment and Path Finding* (TAPF) [36, 17, 7, 15, 3, 32]. Nevertheless, this line of work is mostly restricted to offline scenarios, i.e., tasks (and/or their release times) are assumed to be known. In practice, for example in a sorting center, orders may come dynamically in real-time.

Besides, we draw attention to two seemingly minor but indeed fundamental aspects. **For one**, the robots are usually abstracted to agents doing unit-cost unit-distance cardinal actions, i.e., {`stop`, ↑,

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 167–187, doi:10.4204/EPTCS.439.12

© Fengming Zhu, Weijia Xu, Yifei Guo & Fangzhen Lin
This work is licensed under the
Creative Commons Attribution License.

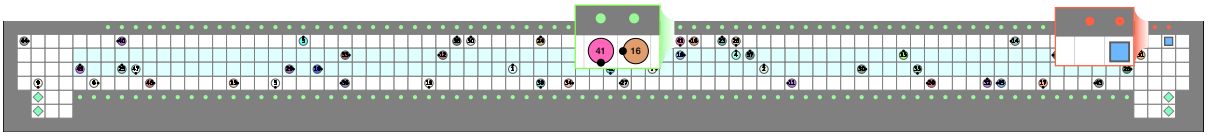


Figure 1: A *non-well-formed* instance currently deployed in Meituan warehouses. The white cells near GREEN dots are delivery ports, while the ones near RED dots are pickup ports. Colored circles heading to different directions with numbers are agents. The colored box (blue) is a pickup port currently assigned to the agent in the same color (ID 45 in the lower right area). Congestion happens a lot near the pickup ports.

$\downarrow$, $\leftarrow$, $\rightarrow$}, what we term as the Type$\oplus$ robot model. The planned paths are later post-processed to executable motions regarding kinematic constraints [9] and action dependencies [8], as a real-world robot has to rotate before going in a different direction. Imaginably, when the rotational cost is not negligible compared to the translational cost, the quality of the plans computed for the Type$\oplus$ robot model will be largely compromised when instantiated to motions. A candidate solution is to revisit and reimplement the existing algorithms over an alternative set of atomic actions {stop, forward, $\circlearrowleft$90, $\circlearrowright$90}, which we advocate in this paper as the Type$\odot$ robot model [2, 37, 12]. **For another**, most of the literature assumes the problem instance to be *well-formed* [18, 15, 34] to guarantee completeness of their methods, which is actually a strong condition requiring that every agent can find a collision-free path to her current goal even if the others are stationary. However, this assumption is often not met in modern warehouses. In particular, the instance (Figure 1) that we consider in this work does not satisfy this condition.

Considering the aforementioned issues, we introduce a formal framework to study the combined problem that organically integrates task assignment and path finding in an online manner. To solve the formalized problem, we **first** develop lifelong path finding algorithms directly for the Type$\odot$ robot model (assuming an arbitrary task assigner), including those adapted from the existing literature and our new rule-based planner which performs both efficiently and effectively, even for non-well-formed instances. **Secondly**, we propose a novel formulation that addresses the online problem of task assignment as optimally solving a Markov Decision Process (MDP), with path planners as hyper-parameters. Due to the complex state space and transition of the formulated MDP, we resort to approximated solutions by reinforcement learning (RL), as well as other non-trivial rule-based ones with insightful observations. *Simulation experiments on warehouse scenarios at Meituan, one of the largest shopping platforms in China, have shown that our system (1) takes only 83.77% of the execution time needed for the currently deployed system at Meituan, outperforming other SOTA algorithms by 8.09%; and (2) can achieve the same throughput with only 60% of the agents of the current scale.* We also draw an important lesson from this study that both path finding and task assignment should fully exploit the warehouse layout, as it is normally fixed in a relatively long period of time after deployment, though the number of agents may still vary. To this end, it might be more worthwhile to investigate layout-dependent-agent-independent solutions instead of entirely general-purpose solutions.

This paper is organized as follows. We first list a few related areas in Section 2. Then the problem formulation is provided in Section 3. We later present our system in two parts: the path planners in Section 4, and the task assigners in Section 5. Experimental results are shown in Section 6, mainly conducted for Meituan warehouse scenarios with various scales of agents. We conclude the paper with a few insightful discussions in Section 7.

2 Related Work

Path Finding. The study of MAPF aims to develop centralized planning algorithms. In spite of the computational complexity being NP-hard in general [35], researchers have developed practically fast planners that can even solve instances with hundreds of agents within seconds. Exemplars can be found via reduction to logic programs [5], prioritized planning [27, 16, 22], conflict-based search [26, 13], depth-first search [20], etc. Most of them can be extended to the online version of the problem, i.e. MAPD, where the goals assigned to agents are priorly unknown [18, 30, 14].

Task Assignment. The earliest attempt is the formulation of *Anonymous*-MAPF (AMAPF) that does not specify the exact goal that an agent must go to [28]. Compared with the labeled version, AMAPF can be solved in polynomial time, via reduction to max-flow problems [36], or target swaps [21]. As a generalization, TAPF explicitly associates each agent with a team [17], or with a set of candidate goals [7, 32], and eventually computes a set of collision-free paths as well as the corresponding assignment matrix. Another analogous formulation is called *Multi-Goal* (MG-)MAPF [29, 31] and its lifelong variant MG-MAPD [34], which also associates each agent with a set of goals, but the visiting order is pre-specified.

We also append some discussion on other less related areas to Appendix B. Despite the rich literature, none of the above directly solves the online problem that a real-world automated warehouse is faced with, which well motivates this work.

3 Problem Definition

We consider a set of agents N, moving along a 4-neighbor grid map given as an undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of unit-cost edges. Let $P, D \subseteq V$ denote the set of pickup ports and the set of delivery ports, respectively. Usually, these two sets are disjoint and are specified alongside the map graph.

Let k starting from 0 denote any arbitrary timestep (to be distinguished from the later notations of tasks). At each timestep k, the local *agent-state* of agent i, denoted as ϕ_i^k, is a 3-tuple consisting of her current location $l_i^k \in V$, direction $d_i^k \in \{n, s, w, e\}$, and goal $g_i^k \in P \cup D$. Φ_i is the set of all possible local states of agent i, and consequently $\Phi = \prod_{i \in N} \Phi_i$ is the set of all possible *joint agent-states*. Each agent is associated with a set of four unit-cost actions $A = \{\texttt{stop}, \texttt{forward}, \circlearrowleft\texttt{90}, \circlearrowright\texttt{90}\}$ (called the $\texttt{Type}\odot$ robot model), with their usual meaning specified using the deterministic function *move*. For example, $move(((3,28), w), \circlearrowleft\texttt{90}) \to ((3,28), n)$, and $move(((3,28), e), \texttt{forward}) \to ((3,29), e)$. While planning paths for agents, we need to avoid the following two types of collisions [28] for any pair of agents (i, j): 1) vertex conflict, i.e., $l_i^k = l_j^k$; and 2) edge conflict, i.e., $l_i^k = l_j^{k+1} \wedge l_j^k = l_i^{k+1}$.

The so-called tasks are composed of a sequence I of typed items. Each item $\iota \in I$ is associated with a type $t \in T$. A back-end demand database specifies for each type a subset of delivery ports that items of the type need to be delivered to. Here we model such a database as a lookup table $L : T \mapsto 2^D$. As L will be changed in real-time, we also let L^k denote the demand database at timestep k. When an agent has finished her last delivery job and returned to a pickup port at timestep k, an item, say of type t, will be loaded onto this idle agent. The system will then check the lookup table L^k, choose one target delivery port $g \in L^k(t)$ to assign to this agent, and delete this demand, i.e. $L^{k+1}(t) = L^k(t) \backslash \{g\}$.

Also, we have an assignment table η that keeps track of which one of the loaded items is assigned to which agent, i.e., $\eta(\iota) = i$ means the item ι is currently carried by agent i. An item ι is *delivered* if there exists a timestep k such that $l_{\eta(\iota)}^k = g_{\eta(\iota)}^k$, i.e., the agent carrying this item has reached her goal. Upon successful delivery, the item ι will be deleted from the entry of η. As η is being changed over time, we

also use the time-indexed version η^t.

We assume (1) L has recorded the demands of a long enough period of time, and therefore, will not be enlarged; and (2) an item will be appended to the system only when it is loaded onto an agent.

With the above notations, we formally define the dynamics of the whole system as a deterministic *system-transition* function over *system-states*.

1. A *system-state* ψ is a tuple consisting of the joint agent-state $\phi = \{\phi_i\}_{i \in N}$, the lookup table L, and the assignment table η at that moment. Let Ψ denote all possible system states. Among them, there is an initial system-state $\psi^0 = (\phi_1^0, \cdots, \phi_N^0, L^0, \eta^0)$.
2. The space of *system-actions* is $\Omega = (A \cup P \cup D)^N$. That is, a *system-action* $\omega \in \Omega$ is an ordered tuple of the atomic actions of agents where any of them can be substituted by an assignment decision. A system-action ω is *executable* under a legal system-state ψ iff

$$\forall i \in N.\ [l_i \in V \backslash (P \cup D) \wedge \omega_i \in A] \vee [l_i \in P \wedge \omega_i \in D] \vee [l_i \in D \wedge \omega_i \in P]$$

3. $\Gamma : \Psi \times \Omega \mapsto \Psi$ is the *system-transition* function, which means (1) if no agents are at the pick-up/delivery ports, then the system proceeds by deterministically moving agents by their reported actions, which will not change the goal component g_i in each agent-state ϕ_i; or (2) if any agent arrives at any pickup (resp. delivery) port, then the system needs to re-assign the agent the next delivery (resp. pickup) port, which will change the goal of that particular agent to the corresponding new location and temporarily force her to stay in-place, and change the demand table L as well as the assignment table η accordingly.

An additional minor assumption is, even if two agents arrive at two different pickup (resp. delivery) ports simultaneously, they will eventually get assigned certain new ports within the next one single timestep one by one in a random order. We do not care about the case where one is waiting for a new-delivery assignment while another is waiting for a new-pickup assignment.

In summary, a *problem instance* is a following tuple $\langle N, G, P, D, A, I, L \rangle$, and consequently a *principle solution* is threefold:

1. $\pi_N : \Phi \mapsto A^N$ is the routing policy that outputs the next action for each agent given their current states. It is unnecessary to compute the entire π_N completely upfront, instead, execution can be interleaved with replanning.
2. $\pi_D : \Psi \times 2^D \mapsto D$ is the delivery selection policy which assigns an agent a delivery port among the candidates according to $L(t)$ when she is at one of the pickup ports and given an item of type t.
3. $\pi_P : \Psi \mapsto P$ is the pickup selection policy which assigns an agent a pickup port to return to when she has finished the delivery.

Note that (1) both π_D and π_P assign one new goal at a time, as we assumed before; (2) both π_D and π_P will change the goal of that particular agent to the corresponding port, while π_N will not; (3) if an agent is at a pickup or delivery port, then her agent-action, even if specified by π_N, will be overwritten to `stop` by the decision of π_D or π_P.

Definition 1 (Feasibility)**.** *Given any system-state ψ and the system-transition Γ, the application of a solution policy (π_N, π_D, π_P) deterministically outputs a successor system-state ψ'. If there is no aforementioned collision between ψ and ψ', then (π_N, π_D, π_P) is a feasible solution.*

We finally define *makespan* as our evaluation metric.

Definition 2 (Makespan)**.** *Given a problem instance with its initial system-state ψ^0, and a feasible solution (π_N, π_D, π_P), an execution trajectory will be generated by the sequential applications of the solution policy $\{\psi^0, \psi^1, \cdots\}$. The makespan is the minimum timestep k such that every item in I is delivered at ψ^k.*

However, in real-world warehouses, the pickup ports are usually concentrated in a restricted local area for operational convenience, e.g., in the top right corner of Figure 1. Therefore, π_P is normally implemented for the purpose of balancing the loads over all pickup ports. **In this work, we merely aim at a *practical solution* consisting of only (π_N, π_D), assuming π_P is given and is not part of the desired solution.**

Example 1 (System pipeline)**.** *As shown in Figure 1,* $\text{ROBOT}_1 \sim \text{ROBOT}_{50}$ *initially rest in random locations after the last system execution. Once the system is launched, each robot moves towards the pickup ports,* RED_1 *and* RED_2*. When* ROBOT_{31} *reaches* RED_2*, the human operator loads a dozen eggs onto it. The system checks the demand database and finds three orders for a dozen eggs, with delivery ports* GREEN_{69}*,* GREEN_{142}*, and* GREEN_{83}*. After consideration, the system decides to send* ROBOT_{31} *to* GREEN_{83} *this time, planning a path while avoiding potential collisions, with subsequent deliveries to the other two ports.*

One may see potential connections between our problem and the standard formulation MAPD in the existing literature, *we postpone some remarks elaborating on the differences to Appendix A, due to the page limit*. We clarify that in the rest of the paper, by "path finding" we mean to compute π_N, and by "task assignment" we mean to compute π_D.

4 Path Finding

In this section, we first review several representative algorithms that can plan collision-free paths in a lifelong fashion. However, they are not always effective for resolving collisions under the `Type`$\odot$ robot model for non-well-formed instances like Figure 1. To this end, we propose a simple-yet-powerful rule-based planner which is capable of efficiently and robustly moving robots without collisions or deadlocks.

4.1 Existing Lifelong Path Finding Algorithms

Prioritized Planning. A straightforward way is to prioritize path finding for each agent by assigning them distinct priorities, known as *Cooperative A** (CA*) [27], which can also be extended to lifelong situations. In descending order of priority, the agents will plan their paths one by one. Once an agent with a higher priority has found her path, those $(state, time)$ pairs along the path will be *reserved* for this particular agent. All subsequent agents with lower priorities will view those reservations as states that are unreachable at the corresponding timesteps, i.e. as spatio-temporal obstacles. Therefore, each agent will need to conduct optimal search over the joint space of state and time. Understandably, there is a chance that an agent with a lower priority cannot compute any feasible path given the preceding path computed by a higher-priority, which makes the algorithm itself incomplete. This situation is even worse under the `Type`$\odot$ robot model, as an agent often needs to rotate in-place before going to an adjacent vertex, which adds extra difficulty of avoiding collisions. *An illustrative example is provided in Appendix C.*

Rolling Horizon Collision Resolution. A more systematic approach for lifelong path finding is to *window* the search process [27]. This idea is further developed by [14] as the *Rolling Horizon Collision Resolution* (RHCR) framework. The framework takes two use-specified parameters: (1) the replanning frequency h and (2) the length of the collision resolution window $w \geq h$, ensuring that no collisions occur

within the next w timesteps. The framework is general enough to encompass most MAPF algorithms. An example is to extend conflict-based search (CBS) [26, 13] to the lifelong version using this RHCR framework, where the high-level constraint tree is expanded only if there are still collisions within the first w timesteps, resulting in a much smaller constraint tree. However, under the `Type`$\odot$ robot model, neighboring agents often require more timesteps to resolve collisions, especially in crowded local areas. *An example is provided in Appendix C.*

4.2 Our Rule-Based Solution: Touring With Early Exit

Instead of doing deliberate planning, we here present a simple-yet-effective rule-based planner, named *Touring With Early Exit* (later denoted as **Touring** for short). As summarized in Algorithm 1, this planner consists of three main rules RULE1-TOURING, RULE2-EARLY-EXIT, and RULE3-SAFE. We will explain them one by one.

Algorithm 1 Touring with early exit

Input: $states = (\{l_i\}_{i\in N}, \{d_i\}_{i\in N}, \{g_i\}_{i\in N})$
Parameter: for any arbitrary timestep k, omitted below
Output: next joint-action $actions$

1: $actions \leftarrow$ RULE1-TOURING$(states)$
2: **if** EXISTS-DEADLOCK$(states)$ **then**
3: $\quad actions \leftarrow$ RULE3-SAFE$(states, actions)$
4: $\quad$ **return** actions
5: **end if**
6: $actions \leftarrow$ RULE2-EARLY-EXIT$(states, actions)$
7: $actions \leftarrow$ RULE3-SAFE$(states, actions)$
8: **return** actions

Firstly, for RULE1-TOURING, we partition the graph into disjoint *Hamiltonian circuits* (HCs), termed as tours $\{\tau_p\}_{p\in P}$, ensuring that each tour covers distinct pickup ports. Let $V_\tau \subset V$ denote the vertices in τ, we have

$$\left(\forall p_1, p_2 \in P.\ p_1 \in \tau_1 \wedge p_2 \in \tau_2 \wedge \tau_1 \neq \tau_2 \iff p_1 \neq p_2\right) \wedge \bigcup_{p\in P} V_{\tau_p} = V \wedge \bigcap_{p\in P} V_{\tau_p} = \emptyset$$

RULE1-TOURING further specifies the fixed direction along which agents will traverse the tour regardless of their goal locations, i.e. blind touring. Figure 2(a) shows an example with two tours (in dashed orange), one covering F2 clockwise and the other covering G2 counter-clockwise. An agent may need more than one action at certain cells for touring, e.g., need a ↻`90` followed by a `forward` at the corner A4.

Secondly, for each tour τ, RULE2-EARLY-EXIT marks certain vertices as turnings, where an agent currently in τ can exit the tour. The set of turnings is denoted as $V_\tau^{turn} \subseteq V_\tau$. An agent i can *exit* her tour τ if she is at the turning that is the closest to her goal by choosing the next action of her shortest plan towards the goal, or continue touring otherwise. Note that it may not be the case that $g_i \in V_\tau$, which may require agents to go across tours. An exiting action is prioritized over a touring action. Two agents who are exiting their own tours simultaneously but moving towards each other will be prioritized by their IDs: the one with the larger ID will exit, while the other continues touring until reaching the next second-best turning. The blue cells in Figure 2 represent the turnings of the respective tours, with 2(b) and 2(c)

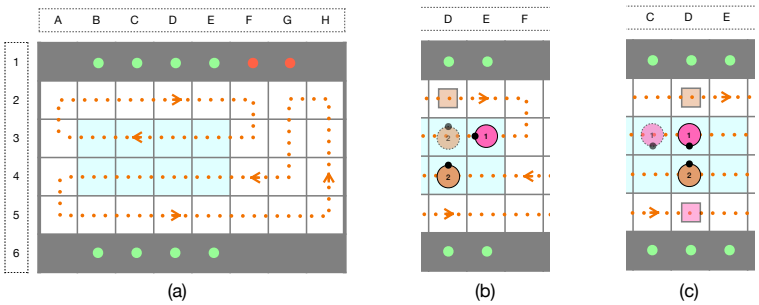


Figure 2: RULE1-TOURING (a) and two prioritized cases (b, c). Colored boxes are the goals.

illustrating the two aforementioned prioritized cases. These turnings can be either user-specified, or searched in terms of minimizing the makespan. *We provide an illustration of how the frequency of the turnings affects the eventual makespan in Appendix E.*

Finally, RULE3-SAFE is applied to revise those actions to collision-free ones. For example, if a preceding agent is rotating, the following agent should not move `forward`; otherwise, collisions may occur. Thus, we design RULE3-SAFE conservatively: for each agent i (1) she observes her adjacent agents but assumes their actions specified by the prior rules may or may not be executed successfully, (2) for either outcome, she checks whether her next action, if it is `forward`, will lead to a collision, (3) if any potential collision is detected, she revises her action to `stop`. Intuitively, this ensures that every agent maintains a safe distance from one another. *In fact, this conservative rule also prevents following-collisions* [28], which will not be discussed in this paper.

Last but not least, one may notice that if there is a subset of agents forming a cycle where each one is about to go to the next location that is currently occupied by another agent in the cycle, RULE3-SAFE will overwrite the actions of all involved agents to `stop`, resulting in a deadlock. Since the planner consisting of only the three main rules is merely a one-step reactive planner, the identical planning step will repeat indefinitely once a deadlock is formed. Therefore, additional inspections need to be made (Line 2 in Algorithm 1), within which a depth-first search is conducted to see if any cycles (and thus the deadlock) exist. Once a deadlock is found, all the *exiting* agents will be interrupted and resume *touring*. By DFS for deadlock detection, we mean to construct a graph at each timestep, termed as $G_d = (V_d, E_d)$. V_d is the set of all agents, and $(i, j) \in E_d$ if i, j are two adjacent agents and $move((l_i, d_i), \texttt{forward}) = (l_j, any)$. If a cycle exists in G_d (mostly caused by RULE2-EARLY-EXIT), the by RULE3-SAFE, it is likely that every agent's next action will be overwritten to `stop`. In fact, **the existence of a cycle in G_d is a necessary condition for the occurrence of a deadlock.** Once a cycle is detected, the whole system will switch to a safe touring-only mode by disabling RULE2-EARLY-EXIT in the next step, thereby breaking the deadlock.

Our **Touring** planner is sound as it eliminates potential collisions by implementing safety rules and avoiding deadlocks in real-time, but it might not be complete. To ensure completeness, every item needs to be finally delivered to its destination: 1) an agent will eventually deliver her item if the delivery port is in the current tour, even in a touring-only mode; 2) if she has missed the current turning to go across tours, she has to seek another in the next few steps, or even the next touring round. In principle, an agent may keep missing chances to go across tours, but it turns out to be a rare case. In practice, an item is usually delivered in its first tour as we found in our experiments.

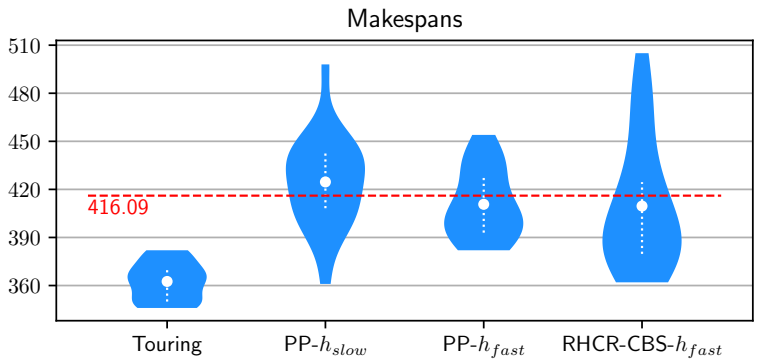


Figure 3: The tested makespans of lifelong path finding algorithms in 50-agent Meituan warehouse scenarios. Dotted lines represent the 25-/75-quantiles, and white dots are the means. The means correspond to the leftmost column of the 50-agent scenario in Table 1. 416.09 is the reference makespan under Meituan's current system.

4.3 Comparison for Path Finding Algorithms

Before adding task assignment to the context, here we first conduct a brief comparison among the above path finding algorithms, assuming a sequence of goals arrives online. We implement the lifelong CA* as a baseline for prioritized planning (denoted as **PP**), and CBS under the RHCR framework with $h = 1, w = 5$ as a baseline for windowed search (denoted as **RHCR-CBS**). We also implement two heuristics for the underlying single-agent search, namely h_{slow}, which merely computes the Manhattan distance between the current location and the goal, and h_{fast}, which additionally counts the minimum number of ↻90/↺90 needed. Hence, here we have $2 \times 2 = 4$ combined baselines. *We report the computing time, even for various scales, in Appendix D.* It turns out **RHCR-CBS-**h_{slow} is too costly for a multi-run evaluation.

As we mentioned, our testing environment (Figure 1) may not be well-formed. **PP** may fail due to improper priority orderings. **RHCR-CBS** may also take a long time for collision resolution, especially when there is a traffic jam near the pickup ports. We treat a replanning of **RHCR-CBS** as failure if the number of leaf nodes in the high-level constraint tree exceeds 50, indicating severe congestion. Once these two methods fail, they will be temporarily switched to **Touring**, and later be switched back.

In Figure 3, we present the entire distributions of the tested makespans over multiple runs. As one can clearly see that our **Touring** planner largely outperforms the other three, and the computing time is entirely negligible compared to the others, as reported in Appendix D. Besides, **RHCR-CBS** outperforms **PP** in most cases, though the average performances are close, as it poorly handles some extreme cases.

5 Task Assignment

In the offline setting where tasks are known *a priori* the assignment problem is well-studied [17, 7, 15, 32], where the combinatorial search of the best task assignment is coupled with path finding. However, when it comes to the online setting, it seems that the best approach so far is to greedily assign tasks at each decision point [18, 22], i.e., to pick up the task so as to minimize the path costs from the current location to starting location of the task. Projecting onto our settings, a greedy assignment is to select among those candidates the delivery port that is closest to the current location. However, no evidence has witnessed that greedy assignments are rational and effective, given the fact that forthcoming tasks are totally unknown. To this end, we extend this greedy strategy into a broader class of strategies, divided into three categories (1) stateless assignment, (2) adaptive assignment, and (3) predictive assignment.

5.1 Stateless Assignment.

As shown in Algorithm 2, MEASUREFUNC is a user-specified function that encodes a measure between the location of the current agent waiting for assignment and the candidate delivery ports. Straightforward options are:

1. **Shortest distances**. This reduces to the greedy strategies that select the closest delivery port.
2. **Negative shortest distances**. This is equivalent to selecting the farthest delivery port. It is usually counter-intuitive, but makes some sense since it may alleviate congestion around the pickup ports, especially when the scale of the agents is large.
3. **Random numbers**. It reduces to random assignments.

Algorithm 2 Stateless assignment

Input: agent i waiting for assignment, item ι of type t, candidate delivery ports $L(t)$
Parameter: any arbitrary timestep k (omitted below)
Output: A selected goal $\in L(t)$

1: **return** $\arg\min_{g \in L(t)} \textsc{MeasureFunc}(g, l_i)$

5.2 Adaptive Assignment.

Stateless assignments do not make the most use of system-state information, e.g., the current locations of all agents. As revealed in Figure 4, the occupation ratio, defined as the fraction of the number of agents over the number of passable cells, of the left half differs significantly from that of the right half. The *closest-first* strategy will inevitably cause high-level congestion around the pickup ports, while *farthest-first* strategy unnecessarily sends agents to farther locations, even though it alleviates traffic jams. The random strategy somehow balances between the former two.

Inspired by this insight, Algorithm 3 further develops an adaptive version, which takes in a congestion threshold α and makes dynamic assignment decisions based on the current state. If there is a heavy traffic in the right half of the map, the system will send agents to farther goals, and similarly otherwise. One can clearly see in Figure 4(d) that the occupation ratio fluctuates more responsively.

The threshold parameter α can be specified by users or searched in terms of minimizing the makespan. *We report comprehensive search results in Figure 7 in Appendix E.*

Algorithm 3 Adaptive assignment

Input: agent i waiting for assignment, item ι of type t, candidate delivery ports $L(t)$, all agents' locations $\{l_i\}_{i \in N}$
Parameter: A threshold α, any timestep k (omitted below)
Output: A selected goal $\in L(t)$

1: $occu_l, occu_r \leftarrow \textsc{OccupationRatio}(\{l_i\}_{i \in N})$
2: **if** $occu_r \leq \alpha$ **then**
3: **return** $\arg\min_{g \in L(t)} \textsc{ShortestDistance}(g, l_i)$
4: **else**
5: **return** $\arg\max_{g \in L(t)} \textsc{ShortestDistance}(g, l_i)$
6: **end if**

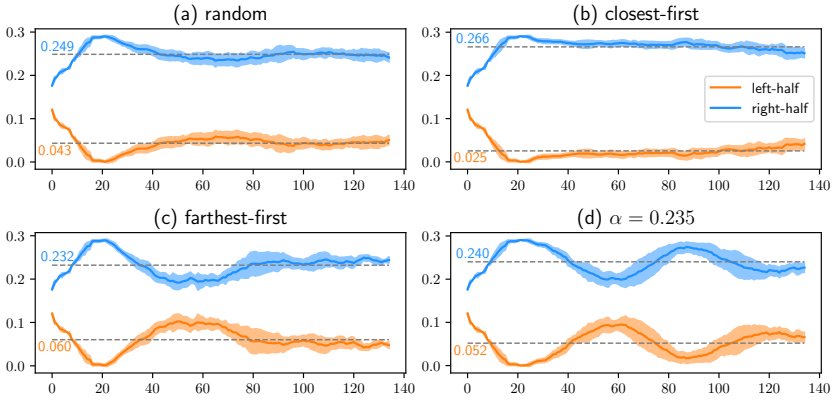


Figure 4: The dynamics of the occupation ratios for different stateless and adaptive assignment strategies in 50-agent cases. Dashed lines represent the means.

5.3 Predictive Assignment.

One can further argue that purely reactive assignments like the above ones do not capture the dynamics of the system. To this end, one needs to make good use of the underlying path finding module which may hint about the dynamic flow of the agent swarm. A systematic way is to formulate the assignment problem as a Markov Decision Process (MDP), taking the path finding module, i.e. the routing policy π_N, as a hyperparameter. The MDP is defined as a 5-tuple $\langle \mathscr{S}, \mathscr{A}, T, R, \gamma \rangle$:

1. **States** $\mathscr{S}$: each $S \in \mathscr{S} \subset \Psi$ is a collection of all system-states where there exists an agent at a pickup port waiting for a new-delivery assignment. We call these *assignment states* to avoid ambiguity.
2. **Actions** $\mathscr{A} = D$: all possible delivery ports. Given a loaded item of type t, the available actions are the delivery ports in $L(t)$.
3. **Transition function** $T : \mathscr{S} \times \mathscr{A} \mapsto \Delta(\mathscr{S})$. Once a new delivery port is assigned to an agent, the system will proceed according to π_N (and the given π_P) until the next assignment state.
4. **Reward function** $R : \mathscr{S} \times \mathscr{A} \times \mathscr{S} \mapsto \mathbb{R}$. The reward is the negative time cost spent between two assignment states. Note that the reward signals are quite "delayed", in the sense that for two adjacent assignment states, the immediate reward received at the latter one might not reflect the delivery cost for the task assigned in the former state (it is usually not delivered yet). Nevertheless, the total accumulated rewards of an episode is indeed the negative makespan to complete the attended item sequence.
5. **Discount factor** $\gamma \in (0, 1)$.

Note that to solve the above MDP is to search for the policy π_D while fixing π_N. By definition, once the optimal policy π_D^* is found, it will instruct the system to assign tasks at any possible assignment state, and therefore, the initial resting locations of agents do not matter. We adopt PPO [25] as the RL algorithm to solve the above MDP, *and will postpone some training details and remarks to Appendix F*.

6 Experimental Results

In this section, we report the main experimental results in Table 1, conducted on Meituan simulated warehouse (Figure 1). The online sequences of items are retrieved from roughly 5-minute segments of

Table 1: Evaluation results. The numbers are the average makespans (↓). The reference makespan is 416.09 by the currently deployed system at Meituan. For each path planner, the result performed by the best task assigner is marked in **bold**. The scenario in teal is the current scale at Meituan. Those in orange represent the best combinations at each scale. Some cells are marked in gray as the RL policies are not explicitly trained for those scenarios. For the adaptive assignment strategies, we report the top-3 ones along with the corresponding thresholds in superscripts. For the RL strategies, besides the best ones of average performances, we also present the best ones of best-case (resp. worst-case) performances, along with the corresponding best-case (resp. worst-case) makespans in superscripts.

(30 agents)	**Random**	**Closest**	**Farthest**	**Adapt**^{1st}	**Adapt**^{2nd}	**Adapt**^{3rd}	**RL**^{avg}	**RL**^{best}	**RL**^{worst}
Touring (ours)	467.00	**412.90**	530.85	$443.45^{0.158}$	$454.15^{0.152}$	$454.15^{0.155}$	425.00	425.00^{412}	425.00^{439}
PP h_{slow}	659.45	**550.75**	678.50	$587.70^{0.158}$	$590.10^{0.149}$	$605.30^{0.146}$	569.45	569.45^{482}	569.45^{762}
PP h_{fast}	641.30	**561.50**	681.50	$589.10^{0.152}$	$589.50^{0.155}$	$610.20^{0.149}$	588.40	588.40^{492}	588.40^{800}
RHCR-CBS h_{fast}	645.00	**539.75**	726.05	$645.20^{0.152}$	$646.20^{0.155}$	$655.10^{0.146}$	641.95	641.95^{495}	641.95^{800}
(40 agents)									
Touring (ours)	392.10	376.30	422.70	$382.40^{0.219}$	$387.30^{0.211}$	$387.30^{0.215}$	**372.05**	372.05^{348}	383.65^{399}
PP h_{slow}	474.50	**427.10**	518.35	$443.50^{0.215}$	$447.20^{0.211}$	$449.05^{0.207}$	452.80	452.80^{425}	473.90^{565}
PP h_{fast}	467.00	**426.70**	516.40	$443.70^{0.219}$	$449.15^{0.215}$	$451.25^{0.211}$	445.20	445.20^{417}	475.45^{535}
RHCR-CBS h_{fast}	463.00	444.95	523.75	$438.85^{0.211}$	$438.85^{0.215}$	$443.10^{0.219}$	**431.55**	431.55^{394}	447.05^{481}
(50 agents)									
Touring (ours)	362.55	358.35	375.15	$\mathbf{348.55}^{0.235}$	$349.35^{0.265}$	$349.80^{0.240}$	350.15	352.70^{316}	350.15^{363}
PP h_{slow}	424.65	392.85	435.45	$\mathbf{388.35}^{0.280}$	$392.65^{0.275}$	$400.55^{0.255}$	409.95	397.10^{359}	409.95^{509}
PP h_{fast}	410.70	**390.15**	434.20	$396.70^{0.280}$	$399.40^{0.265}$	$400.15^{0.260}$	398.25	402.70^{361}	398.25^{444}
RHCR-CBS h_{fast}	409.60	401.00	415.90	$384.25^{0.280}$	$385.20^{0.275}$	$386.10^{0.265}$	384.90	$\mathbf{382.20}^{363}$	384.90^{468}
(60 agents)									
Touring (ours)	350.60	352.50	352.40	$\mathbf{335.50}^{0.281}$	$337.10^{0.293}$	$337.10^{0.299}$	342.70	342.70^{308}	342.70^{359}
PP h_{slow}	390.90	380.45	411.00	$\mathbf{369.35}^{0.287}$	$370.20^{0.293}$	$373.10^{0.329}$	375.10	375.10^{345}	375.10^{403}
PP h_{fast}	394.15	382.80	397.15	$\mathbf{371.75}^{0.299}$	$372.65^{0.329}$	$378.45^{0.311}$	391.05	391.05^{356}	391.05^{499}
RHCR-CBS h_{fast}	372.50	370.00	375.90	$\mathbf{357.35}^{0.305}$	$360.55^{0.287}$	$360.85^{0.323}$	372.85	372.85^{354}	372.85^{469}
(70 agents)									
Touring (ours)	346.45	354.65	344.50	$\mathbf{333.40}^{0.353}$	$333.60^{0.339}$	$334.10^{0.360}$	338.80	338.80^{308}	338.80^{354}
PP h_{slow}	375.95	381.15	393.85	$374.95^{0.325}$	$375.40^{0.388}$	$375.95^{0.304}$	**373.50**	373.50^{347}	373.50^{394}
PP h_{fast}	371.25	372.10	372.10	$\mathbf{364.95}^{0.332}$	$364.95^{0.360}$	$365.35^{0.304}$	390.90	390.90^{340}	390.90^{513}
RHCR-CBS h_{fast}	362.50	377.20	365.55	$\mathbf{351.90}^{0.367}$	$353.30^{0.353}$	$354.10^{0.381}$	362.20	362.20^{337}	362.20^{435}

the system's log containing around 140 items of approximately 50 types, while the demand database L is made from mobile orders made by the customers in a longer period of around 6 hours. The two dimensions of Table 1 represent the choices of path planners and task assigners, respectively. The numbers are the average *makespans* over multiple runs with agents initialized in random locations. In each run, the system is required to deliver a sequence of items of a fixed length that arrives online. In other words, we evaluate the average cost it takes to accomplish the same amount of throughput, for each pair of path planners and task assigners. For the adaptive assignment strategies, we report the top-3 ones along with the corresponding thresholds in superscripts. For the RL strategies, besides the best ones for average performances, we also present the best ones for best-case (resp. worst-case) performances, along with the corresponding best-case (resp. worst-case) makespans in superscripts. We only train RL policies over the **Touring** planner, since the others are not fast enough and will take a tremendous amount of time for RL training. However, we can slightly abuse a trained assignment policy by testing it with the other three path planners as the state spaces are same.

As an overview, our **Touring** planner outperforms the other three regardless of the task assigner. As for the task assigner, (1) when the number of agents is ≥ 50, adaptive strategies are surprisingly effective, even slightly better than RL ones, and the *closest-first* strategy is not necessarily better than the *farthest-first* strategy. (2) when the number of agents is < 50, it might be redundant to use adaptive strategies as the density of agents is quite low; instead, stateless ones or RL ones are better choices. Although RL strategies can achieve comparable performances in practice (even optimal performance in theory if trained well), it depends on the user whether the training cost is a worthwhile effort.

Looking closer, we point out two insights:

1. *Time efficiency*. Regarding the current scale of Meituan (50 agents), our system only needs 83.77% of the makespan to deliver the same amount of throughput, compared to the company's current system $^{(348.55/416.09)}$, outperforming the best$^{(382.20/416.09)}$ among the rest by 8.09%.

2. *Economic efficiency*. Note that there is a continuing improvement$^{(348.55\rightarrow335.50)}$ while increasing the number of agents to 60. However, the marginal gain of further increasing to 70 agents is negligible$^{(335.50\rightarrow333.40)}$. In fact, only 30 agents can fulfill the current throughput with even slightly shorter time$^{(412.90)}$, resulting in a 40% reduction in fixed costs for purchasing robots.

We additionally point out that, while the *closer-first* strategy seems effective in the experiments and is widely adopted in the literature, such fixed-priority methods are not desirable in practice. Similar to processor scheduling in OS, a significant drawback is that certain tasks may continually be preempted, i.e., a delivery port gets no item for a long operational period.

7 Conclusion and Future Work

In this paper, we conduct a case study on the real-world problem of warehouse automation by combining lifelong multi-robot path finding and dynamic task assignment in an online fashion. As a result, we manage to speed up package delivery given the current scale at Meituan, and also identify potential profitable upgrades of the system. An important lesson from this study is that given the layout of the warehouse, once deployed, is normally fixed in a relatively long period of time, it is worthwhile to have both the routing module and the assignment module that take advantage of the layout. However, both modules should be general enough to account for the varying number of robots available. We conclude the paper by clarifying some concerns for generalizability and pointing out some future work.

Generalizability and Limitation. *For the routing module*, in this study, we identify two disjoint HCs that cover all delivery ports, with each circuit servicing one pickup port. Additional rules are implemented to prioritize agents crossing between the tours. While a similar strategy can be applied in general, determining the optimal approach may be nontrivial, and thus an intriguing topic for future research. For instance, one needs to consider the trade-off between using multiple HCs to cover different pickup ports versus employing one single large HC that covers all delivery and pickup ports. *For the assignment module*, 1) for stateless assignments, one can replace the distance measure with another suitable one; 2) for adaptive assignments, one can divide the whole map into a few subareas to monitor congestion (we presented the case of two monitored subareas), and 3) for predictive assignments, the MDP formulation is layout-independent, solely taking in a given pathfinder. A limitation of this work is that we search an assignment policy w.r.t. a fixed routing policy, which is an open-loop control. The next step is to couple the search of these two, though it will be computationally challenging.

Acknowledgments

This work was supported in part by Meituan Academy of Robotics Shenzhen (MARS). We thank our colleagues from Meituan—Yuan Liu, Shumin Zhang, Fuqiang Hei, and Yuxuan Zhang—for their invaluable assistance and collaboration in this project, and colleagues from HKUST—Yuxin Pan, Yangfan Wu and Xiaomeng Zhu—for their insightful feedback during the revision of this paper. We also appreciate the constructive comments from the anonymous reviewers.

References

[1] Kris Braekers, Katrien Ramaekers & Inneke Van Nieuwenhuyse (2016): *The vehicle routing problem: State of the art classification and review*. Computers & industrial engineering 99, pp. 300–313. Available at https://doi.org/10.1016/j.cie.2015.12.007.

[2] Shao-Hung Chan, Zhe Chen, Teng Guo, Han Zhang, Yue Zhang, Daniel Harabor, Sven Koenig, Cathy Wu & Jingjin Yu (2024): *The League of Robot Runners Competition: Goals, Designs, and Implementation*. In: ICAPS 2024 System's Demonstration track.

[3] Zhe Chen, Javier Alonso-Mora, Xiaoshan Bai, Daniel D Harabor & Peter J Stuckey (2021): *Integrated task assignment and path planning for capacitated multi-agent pickup and delivery*. IEEE Robotics and Automation Letters 6(3), pp. 5816–5823, doi:10.1109/LRA.2021.3074883.

[4] Mehul Damani, Zhiyao Luo, Emerson Wenzel & Guillaume Sartoretti (2021): *PRIMAL$_2$: Pathfinding via reinforcement and imitation multi-agent learning-lifelong*. IEEE Robotics and Automation Letters 6(2), pp. 2666–2673, doi:10.1109/LRA.2021.3062803.

[5] Esra Erdem, Doga Kisa, Umut Oztok & Peter Schüller (2013): *A general formal framework for pathfinding problems with multiple agents*. In: Proceedings of the AAAI Conference on Artificial Intelligence, 27, pp. 290–296. Available at https://doi.org/10.1609/aaai.v27i1.8592.

[6] Matthew L Ginsberg (1989): *Universal planning: An (almost) universally bad idea*. AI magazine 10(4), pp. 40–40. Available at https://doi.org/10.1609/aimag.v10i4.964.

[7] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W Durham & Nora Ayanian (2018): *Conflict-Based Search with Optimal Task Assignment*. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, pp. 757–765. Available at https://dl.acm.org/doi/abs/10.5555/3237383.3237495.

[8] Wolfgang Hönig, Scott Kiesel, Andrew Tinka, Joseph W Durham & Nora Ayanian (2019): *Persistent and robust execution of MAPF schedules in warehouses*. IEEE Robotics and Automation Letters 4(2), pp. 1125–1131, doi:10.1109/LRA.2019.2894217.

[9] Wolfgang Hönig, TK Kumar, Liron Cohen, Hang Ma, Hong Xu, Nora Ayanian & Sven Koenig (2016): *Multi-agent path finding with kinematic constraints*. In: Proceedings of the International Conference on Automated Planning and Scheduling, 26, pp. 477–485. Available at https://doi.org/10.1609/icaps.v26i1.13796.

[10] Shengyi Huang & Santiago Ontañón (2022): *A Closer Look at Invalid Action Masking in Policy Gradient Algorithms*. In: The International FLAIRS Conference Proceedings, 35. Available at https://doi.org/10.32473/flairs.v35i.130584.

[11] Anant Singh Jain & Sheik Meeran (1999): *Deterministic job-shop scheduling: Past, present and future*. European journal of operational research 113(2), pp. 390–434. Available at https://doi.org/10.1016/S0377-2217(98)00113-1.

[12] He Jiang, Yulun Zhang, Rishi Veerapaneni & Jiaoyang Li (2024): *Scaling lifelong multi-agent path finding to more realistic settings: Research challenges and opportunities*. In: Proceedings of the International Sym-

posium on Combinatorial Search, 17, pp. 234–242. Available at https://doi.org/10.1609/socs.v17i1.31565.

[13] Jiaoyang Li, Wheeler Ruml & Sven Koenig (2021): *Eecbs: A bounded-suboptimal search for multi-agent path finding*. In: *Proceedings of the AAAI conference on artificial intelligence*, 35, pp. 12353–12362. Available at https://doi.org/10.1609/aaai.v35i14.17466.

[14] Jiaoyang Li, Andrew Tinka, Scott Kiesel, Joseph W Durham, TK Satish Kumar & Sven Koenig (2021): *Lifelong multi-agent path finding in large-scale warehouses*. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 35, pp. 11272–11281. Available at https://doi.org/10.1609/aaai.v35i13.17344.

[15] Minghua Liu, Hang Ma, Jiaoyang Li & Sven Koenig (2019): *Task and path planning for multi-agent pickup and delivery*. In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. Available at https://dl.acm.org/doi/abs/10.5555/3306127.3331816.

[16] Hang Ma, Daniel Harabor, Peter J Stuckey, Jiaoyang Li & Sven Koenig (2019): *Searching with consistent prioritization for multi-agent path finding*. In: *Proceedings of the AAAI conference on artificial intelligence*, 33, pp. 7643–7650. Available at https://doi.org/10.1609/aaai.v33i01.33017643.

[17] Hang Ma & Sven Koenig (2016): *Optimal Target Assignment and Path Finding for Teams of Agents*. In: *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pp. 1144–1152. Available at https://dl.acm.org/doi/abs/10.5555/2936924.2937092.

[18] Hang Ma, Jiaoyang Li, TK Satish Kumar & Sven Koenig (2017): *Lifelong Multi-Agent Path Finding for Online Pickup and Delivery Tasks*. In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pp. 837–845. Available at https://dl.acm.org/doi/10.5555/3091125.3091243.

[19] Alan S Manne (1960): *On the job-shop scheduling problem*. *Operations research* 8(2), pp. 219–223. Available at https://doi.org/10.1287/opre.8.2.219.

[20] Keisuke Okumura (2023): *Lacam: Search-based algorithm for quick multi-agent pathfinding*. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 37, pp. 11655–11662. Available at https://doi.org/10.1609/aaai.v37i10.26377.

[21] Keisuke Okumura & Xavier Défago (2023): *Solving simultaneous target assignment and path planning efficiently with time-independent execution*. *Artificial Intelligence* 321, p. 103946. Available at https://doi.org/10.1016/j.artint.2023.103946.

[22] Keisuke Okumura, Manao Machida, Xavier Défago & Yasumasa Tamura (2022): *Priority inheritance with backtracking for iterative multi-agent path finding*. *Artificial Intelligence* 310, p. 103752. Available at https://doi.org/10.1016/j.artint.2022.103752.

[23] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig & Howie Choset (2019): *Primal: Pathfinding via reinforcement and imitation multi-agent learning*. *IEEE Robotics and Automation Letters* 4(3), pp. 2378–2385, doi:10.1109/LRA.2019.2903261.

[24] Marcel Schoppers (1987): *Universal Plans for Reactive Robots in Unpredictable Environments*. In: *IJCAI*, 87, Citeseer, pp. 1039–1046. Available at https://dl.acm.org/doi/abs/10.5555/1625995.1626091.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford & Oleg Klimov (2017): *Proximal policy optimization algorithms*. *arXiv preprint arXiv:1707.06347*.

[26] Guni Sharon, Roni Stern, Ariel Felner & Nathan R Sturtevant (2015): *Conflict-based search for optimal multi-agent pathfinding*. *Artificial intelligence* 219, pp. 40–66. Available at https://doi.org/10.1016/j.artint.2014.11.006.

[27] David Silver (2005): *Cooperative pathfinding*. In: *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*, 1, pp. 117–122. Available at https://doi.org/10.1609/aiide.v1i1.18726.

[28] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar et al. (2019): *Multi-agent pathfinding: Definitions, variants, and benchmarks*. In: *Proceedings of the International Symposium on Combinatorial Search*, 10, pp. 151–158. Available at https://doi.org/10.1609/socs.v10i1.18510.

[29] Pavel Surynek (2021): *Multi-goal multi-agent path finding via decoupled and integrated goal vertex ordering*. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 35, pp. 12409–12417. Available at https://doi.org/10.1609/aaai.v35i14.17472.

[30] Jiří Švancara, Marek Vlk, Roni Stern, Dor Atzmon & Roman Barták (2019): *Online multi-agent pathfinding*. In: *Proceedings of the AAAI conference on artificial intelligence*, 33, pp. 7732–7739. Available at https://doi.org/10.1609/aaai.v33i01.33017732.

[31] Mingkai Tang, Yuanhang Li, Hongji Liu, Yingbing Chen, Ming Liu & Lujia Wang (2024): *MGCBS: An Optimal and Efficient Algorithm for Solving Multi-Goal Multi-Agent Path Finding Problem*. In Kate Larson, editor: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, International Joint Conferences on Artificial Intelligence Organization, pp. 249–256. Available at https://doi.org/10.24963/ijcai.2024/28. Main Track.

[32] Yimin Tang, Zhongqiang Ren, Jiaoyang Li & Katia Sycara (2023): *Solving multi-agent target assignment and path finding with a single constraint tree*. In: *2023 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, IEEE, pp. 8–14, doi:10.1109/MRS60187.2023.10416794.

[33] Paolo Toth & Daniele Vigo (2014): *Vehicle routing: problems, methods, and applications*. SIAM. Available at https://doi.org/10.1137/1.9781611973594.fm.

[34] Qinghong Xu, Jiaoyang Li, Sven Koenig & Hang Ma (2022): *Multi-goal multi-agent pickup and delivery*. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 9964–9971, doi:10.1109/IROS47612.2022.9981785.

[35] Jingjin Yu & Steven LaValle (2013): *Structure and intractability of optimal multi-robot path planning on graphs*. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 27, pp. 1443–1449. Available at https://doi.org/10.1609/aaai.v27i1.8541.

[36] Jingjin Yu & Steven M LaValle (2013): *Multi-agent path planning and network flow*. In: *Algorithmic Foundations of Robotics X: Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics*, Springer, pp. 157–173. Available at https://doi.org/10.1007/978-3-642-36279-8_10.

[37] Yue Zhang, Daniel Harabor, Pierre Le Bodic & Peter J Stuckey (2023): *Efficient Multi Agent Path Finding with Turn Actions*. In: *Proceedings of the International Symposium on Combinatorial Search*, 16, pp. 119–127. Available at https://doi.org/10.1609/socs.v16i1.27290.

[38] Fengming Zhu & Fangzhen Lin (2025): *Computing Universal Plans for Partially Observable Multi-Agent Routing Using Answer Set Programming*. In: *Proceedings of the International Conference on Logic Programming (ICLP)*, to appear. Available at https://arxiv.org/abs/2305.16203.

A Relation to MAPD

In MAPD, an online task t_i is characterized by a pickup port s_i and a delivery port g_i with a priorly unknown release time. Once an agent becomes idle, she will select one task $t^* = (s^*, g^*)$ of her best interest from the released ones, and then plan a path from her current location to g^* through s^*. Mapping to our settings, an agent becomes idle only when she arrives at a pickup port, and shall then be assigned one delivery port from the candidates, say $\{g_1, \cdots, g_k\}$. Suppose the system will simply pair each delivery port with a pickup port immediately, for which the particular agent will return to after the delivery. Then it is equivalent to, in the language of MAPD, releasing k tasks $\{(g_1, \pi_P(g_1)), \cdots, (g_k, \pi_P(g_k)\}$. However, after choosing one from the k tasks and assigning it to an agent, the rest $(k-1)$ tasks will be temporarily removed, or "deactivated", from the pool of released tasks until the next item of the same type arrives.

B More Discussion on Related Work

The problem presented in this paper pertains to some other important areas in planning and operations research.

Universal Planning. Unlike the idealized one-shot MAPF, fully automating real-world warehouses requires lifelong path finding. However, most of the existing work [18, 30, 14, 34] still focuses on the solution concept as a set of collision-free paths, which is a sequence of joint actions. Such a solution concept is vulnerable if there is any uncertainty, e.g. unknown future goals, or even system contingencies. We argue that one can turn to the solution concept of universal plans [24, 6]. Although universal plans are even harder to compute, there are some exemplars using multi-agent reinforcement learning [23, 4], or via reduction to logic programs [38].

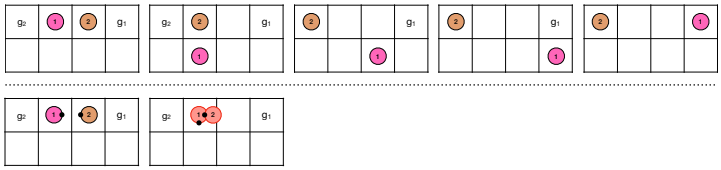
Scheduling. One may also notice the analogy between TAPF and job-shop scheduling problems (JSSP) [19, 11] or vehicle routing problems (VRP) [33, 1]. However, there are at least two key differences: (1) job durations in JSSP and route lengths in VRP are usually known in advance and (2) the execution of jobs or routes is independent of each other. Neither of these two conditions holds in TAPF, especially when the tasks are released online.

C Side Effects of the Type$\odot$ Robot Model

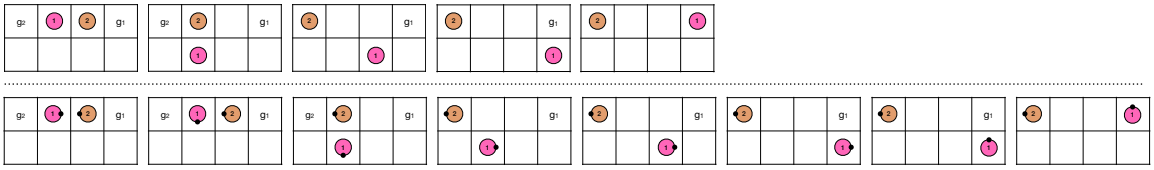
When the state of an agent is lifted from pure locations to (location, direction) pairs, there will be extra difficulty resolving collisions. We here show three examples for (a) *cooperative A** (CA*) [27], (b) *conflict-based search* (CBS) [26], and (c) *priority inheritance with backtracking* (PIBT) [22], respectively.

1. Figure 5(a) shows a case where CA* fails for the Type$\odot$ robot model. Suppose agent 2 is prioritized over agent 1, then agent 1 will move away immediately under the Type$\oplus$ robot model. However, under the Type$\odot$ robot model, agent 1 has to rotate first and thus cannot manage to avoid collision at the very next timestep.
2. Figure 5(b) shows a case where it takes CBS a longer time to resolve collisions under the Type$\odot$ robot model. The main reason is still due to the rotational cost. Similarly for the execution of *priority-based search* (PBS) [16].
3. Figure 5(c) shows a failed case due to deeper theoretical reasons. Instead of performing any best-first search, PIBT repeats one-timestep planning until the terminal state, and therefore, it needs a

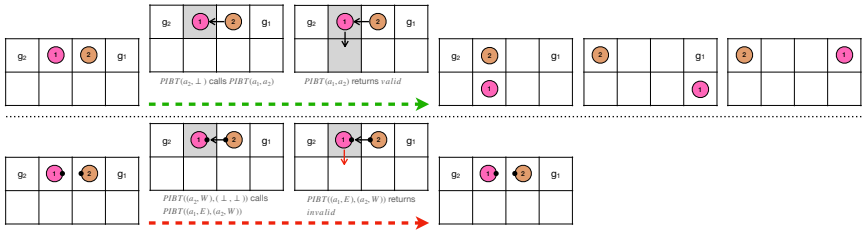
crucial lemma to make sure the total number of execution steps is always bounded (see **Lemma 1** in [22]), i.e., *at each timestep the agent with the highest priority will manage to move one step closer to her goal*. Nevertheless, when the states of each agent are lifted from only locations to (location, direction) pairs, this lemma no longer holds as a counter-example is provided in Figure 5(c).



(a) Cooperative A* may fail for the `Type`$\odot$ robot model



(b) Conflict-based search works but takes more timesteps (same consequence by priority-based search in this particular case).



(c) PIBT fails for the `Type`$\odot$ robot model.

Figure 5: Examples showing the added difficulty of resolving collisions with the `Type`$\odot$ robot model.

D Computing Time

We report the planning time per step in Table 2. Experiments are conducted on a MacBook Air with Apple M2 CPU and 16 GB memory. The planners are all implemented in Python, therefore, those numbers are merely for relative comparisons within this work.

	30	**40**	**50**	**60**	**70**
Touring (ours)	**0.008**	**0.012**	**0.018**	**0.025**	**0.032**
PP h_{fast}	0.066	0.115	0.225	0.500	0.713
PP h_{slow}	0.169	0.287	0.448	0.892	1.028
RHCR-CBS h_{fast}	0.765	0.718	1.842	2.642	2.448
RHCR-CBS h_{slow}	3.023	3.070	7.081	7.821	8.952

Table 2: Planning time per step in seconds, implemented in Python.

E Parameter Search

In the design of both the **Touring** and adaptive task assignment, there are certain hyper-parameters. We here show how the best option is searched in terms of minimizing the eventual makespan.

1. Turning frequency (Figure 6). Figure 1 has presented the extreme where every possible cell that can be a turning is set as a turning, i.e., of frequency 1. One can gradually "sparsify" the turnings to see if the overall makespan gets worse. It turns out, the more turnings you have, the better the makespan on average will be.
2. Adaptive Threshold (Figure 7). As the occupation ratio is defined as the number of agents over the number of passable cells in that part of area, the spectrum of tested thresholds in N-agent scenarios will be considerably less than those in N'-agent scenarios if $N < N'$. One can clearly observe that our **Touring** planner significantly outperforms the other three, and the threshold that makes the lowest box plot is the most desired one. Another observation from Figure 7 is that ours is also more stable than the other three, as the variations (the length of those boxes) are relatively small in most cases.

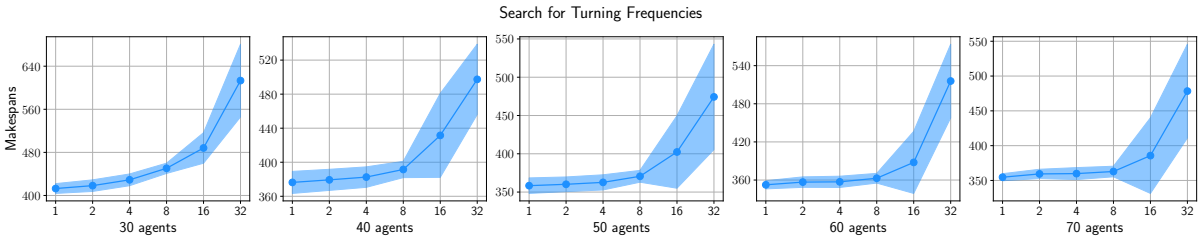


Figure 6: Makespans over different turning frequency in various scales of agents in Meituan warehouse simulation. The X-axis means "there will be a turning every x cells".

F RL Training Details and Remarks

Here we reveal the details of RL training skipped in Section 5.3.

Actions. We directly mask out unavailable actions (those delivery ports that do not need the item) at each assignment state, instead of signaling large negative rewards. In principle, these two are equivalent in terms of the value of the eventual optimal policy, but the former one will guide the policy optimization to converge faster [10].

State features. As defined in Section 5.3, assignment states contain necessary information from system-states. Here we make each state of size $num_of_agents \times (2+1)$, which means to mark each agent's location and direction (converted to [0, 90, 80, 270]). The location feature is further normalized by the layout shape, and the direction feature is normalized by 360.

Episodes. We train the RL agents over one set of item sequences while evaluate it over another set of item sequences.

Hyper-parameters. Both the value network and the policy network are MLPs of size $H \times H \times H \times H$ followed by respective value/policy heads. We attach some training samples in Figure 8, for H chosen from [128, 256, 1024]. The number of total training steps can also be seen in this figure. It turns out networks with $H = 1024$ tend to overfit in most cases.

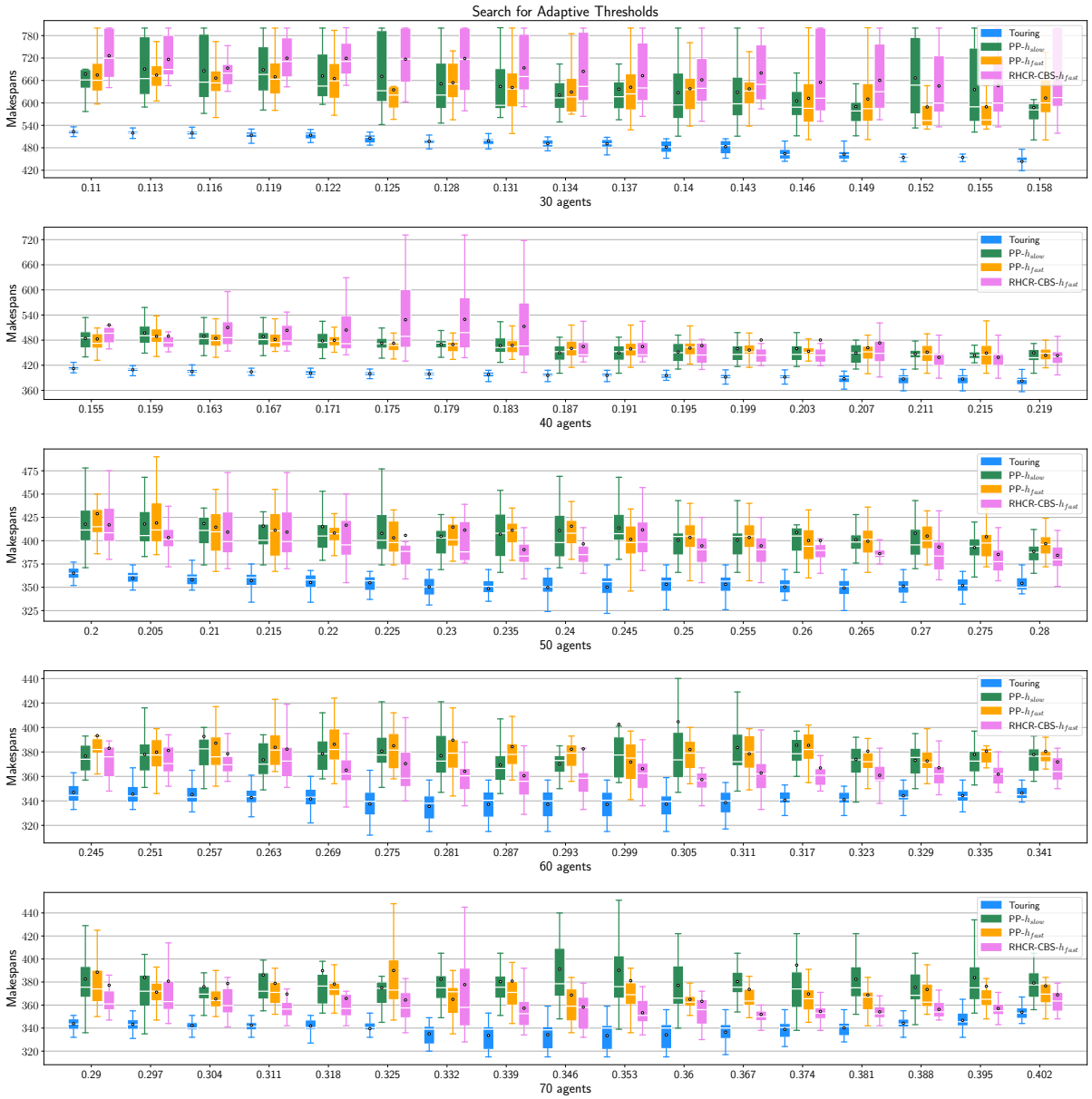


Figure 7: The box-plot of makespans over different adaptive thresholds with various scales of agents in Meituan warehouse simulation.

Necessity of RL. In Table 1, RL performs better than adaptive methods with 30/40 agents, though slightly worse with 50/60/70 agents. Please also note that for the column RL^{best}, the number$^{\text{(in superscripts)}}$ is always the best. RL is used here to solve the underlying MDP, which is a direct characterization of task assignment. This systematic formulation is crucial for a comprehensive study, even though the training outcomes may lag in practice. Additionally, while the closer-first strategy seems effective in the experiments, such fixed-priority methods are not desirable in practice. Similar to processor scheduling in OS, a significant drawback is that certain tasks may continually be preempted, i.e., a delivery port gets no item for a long operational period.

Figure 8: Training processes for different network sizes in various scales of agents.

Delayed rewards. A critical challenge is mentioned in Section 5.3, termed *delayed rewards*. By our definition, the reward is the negative time cost spent between two assignment states. These reward signals are quite "delayed", in the sense that for two adjacent assignment states, the immediate reward received at the latter one might not reflect the delivery cost for the task assigned in the former state (it is usually not delivered yet). This issue is worsened when the number of agents grows, as a longer sequence of transitions is needed for an item to be delivered.

Alternative algorithms. Other than PPO, we have also considered DQN and REINFORCE, expecting REINFORCE to outperform PPO and DQN, as it collects accumulated rewards for the entire episode in a Monte Carlo manner without learning a value model, instead of minimizing the temporal difference error like PPO and DQN. However, PPO performed the best eventually.

G Running Example Recordings

We provide three video clips under the `/figs` folder of the online repository[1] for the *Touring* planner coupled with the *closest-first* assigner, the *adaptive* assigner with $\alpha = 0.235$, and the *RL* assigner with the best best-case performance, respectively.

1. `warehouse_50_touring_closest.mp4` with makespan 355.
2. `warehouse_50_touring_alpha0235.mp4` with makespan 325.
3. `warehouse_50_touring_rl.mp4` with makespan 316.

In all the above instances, the initial states are the same, i.e. the corresponding agents are in the same locations and towards the same directions at the beginning, and the online item sequences are also the same.

[1] `https://github.com/Fernadoo/Online-TAPF`

A framework for Conditional Reasoning in Answer Set Programming

Mario Alviano
DEMACS, University of Calabria, Italy
mario.alviano@unical.it

Laura Giordano Daniele Theseider Dupré
DISIT, University of Piemonte Orientale, Italy
laura.giordano@uniupo.it dtd@uniupo.it

In this paper we introduce a Conditional Answer Set Programming framework (Conditional ASP) for the definition of conditional extensions of Answer Set Programming (ASP). The approach builds on a conditional logic with typicality, and on the combination of a conditional knowledge base with an ASP program, and allows for conditional reasoning over the answer sets of the program. The formalism relies on a multi-preferential semantics, and on the KLM preferential semantics, as a special case. Conditional entailment is encoded in ASP and a complexity upper-bound is provided.

1 Introduction

Preferential logics [15, 31, 33, 27, 34, 28, 7, 9, 26] have been proposed to provide axiomatic foundations of non-monotonic or defeasible reasoning, and to capture commonsense reasoning. They have their roots in conditional logics [30, 32], and allow for conditional statements (simply, *conditionals*) of the form $\alpha \mathrel{|\!\sim} \beta$, meaning that "normally if α holds, β holds".

Answer set programming (ASP) is one of the main computational logic paradigms. It is a formalism for declarative problem solving, allowing for the declarative specification of knowledge through a set of rules. The semantics of an ASP program is given by assigning it a collection of *stable models* [18, 17], also called *answer sets*.

ASP has been used for encoding entailment in some preferential logics, specifically, for preferential extensions of low complexity Description Logics (DLs) of the $\mathscr{EL}^{\bot}$ family [5]. In particular, ASP has been used for reasoning in a ranked concept-wise extension of $\mathscr{EL}^{\bot}$ with typicality [23], for reasoning with many-valued weighted conditional knowledge bases [3, 1], and also for defeasible reasoning based on the Rational Closure [22].

In this paper, we follow a different path. We aim at combining conditional logic and answer set programming by pairing a conditional knowledge base K with an ASP program Π, with the aim of using the conditional logic in the verification of conditional properties of the ASP program. In the preferential semantics, a conditional knowledge base induces a preference relation between worlds (propositional interpretations); when combined with an ASP program Π, the preference relation is induced on the set of *answer sets* of Π, and a preferential interpretation can be constructed from such a set. Verifying *conditional properties of an ASP program*, given the associated conditional KB, is especially of interest when Π has many answer sets, and inspecting them is not viable.

In the literature many different semantics have been considered for conditional knowledge bases, and different closure constructions for strengthening preferential and rational entailment have been explored. Among them are the well known Rational Closure [28] and Lexicographic Closure [29]. They exploit a single preference relation, while, in the paper, we explore an approach based on *multiple preference relations*, which has been considered in conditional description logics with typicality for ranked knowledge bases [23], and for weighted (many-valued) knowledge bases [3, 1].

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 188–201, doi:10.4204/EPTCS.439.13

© M. Alviano, L. Giordano and D. Theseider Dupré
This work is licensed under the
Creative Commons Attribution License.

As in the Propositional Typicality Logic [8] (and in some DLs with typicality [21]), we adopt a language in which conditionals are formalized based on material implication plus the *typicality operator* $\mathbf{T}$. The typicality operator allows for the definition of *conditional implications* $\mathbf{T}(\alpha) \rightarrow \beta$, which correspond to the KLM conditionals $\alpha \mathrel{|\!\sim} \beta$ [27, 28], meaning that "in the typical situations in which α holds, β also holds". For instance, the conditional implication:

$$\mathbf{T}(\mathit{granted_Loan}) \rightarrow \mathit{living_in_town} \wedge \mathit{young}$$

means that, normally, if a loan is granted to a person, she lives in town and is young. The typicality operator allows for more general implications of the form $\alpha \rightarrow \beta$, where $\mathbf{T}$ may occur in both α and β. For instance, the converse implication $\mathit{living_in_town} \wedge \mathit{young} \rightarrow \mathbf{T}(\mathit{granted_Loan})$ means that living in town and being young implies that normally the loan is granted.

The preferential semantics of the logic exploits *multiple preference relations* $<_A$ with respect to different distinguished propositions A, following the approach developed for ranked and weighted KBs in description logics, based on a *multi-preferential semantics* [23, 3] and in some refinements of the KLM logics [19], where preferences are allowed with respect to different aspects.

In the paper, we let a *Conditional ASP (CondASP) program* to be a pair (Π, K), where Π is an ASP program and K is a *defeasible knowledge base*. We consider the special case where K is a weighted KB, in which distinguished propositions A_i are associated with a set of conditionals, each one with a weight, representing the plausibility/implausibility of a defeasible property.

For instance, consider the following example, in which defeasible properties of students and employees are specified through a set of weighted conditionals, as follows:

(d_1) $\mathbf{T}(\mathit{employee}) \rightarrow \mathit{young}$, -50 $\quad$ (d_4) $\mathbf{T}(\mathit{student}) \rightarrow \mathit{young}$, 90

(d_2) $\mathbf{T}(\mathit{employee}) \rightarrow \mathit{has_boss}$, 100 $\quad$ (d_5) $\mathbf{T}(\mathit{student}) \rightarrow \mathit{has_boss}$, -80

(d_3) $\mathbf{T}(\mathit{employee}) \rightarrow \mathit{has_classes}$, -70 $\quad$ (d_6) $\mathbf{T}(\mathit{student}) \rightarrow \mathit{has_classes}$, 80

Conditionals $(d_1 - d_3)$ describe properties of typical employees, who have a boss, and are not likely to be young nor to have classes; $(d_4 - d_6)$ describe properties of typical students. Absolute values of weights correspond to degrees of (im)plausibility. Consider a situation S_1 describing a student and employee who is young, has no boss and has classes, and a situation S_2 describing a student and employee who is not young, has a boss and has no classes. Given the weights above, one would regard S_1 as a more plausible situation for a student than S_2, i.e., $S_1 <_{\mathit{student}} S_2$; and S_2 as a more plausible situation for an employee than S_1, i.e., $S_2 <_{\mathit{employee}} S_1$.

In the paper, after defining (Section 2) a *two-valued and multi-preferential conditional logic with typicality*, which is a generalization of KLM preferential semantics, we develop (Section 3) a construction for defining, for a set of distinguished propositions A_i, preference relations $\leq_{A_i}$, preorders on the answer sets of the program. In general, alternative constructions may be viable for defining the relations $\leq_{A_i}$, depending on the definition of the conditional knowledge base (whether it contains conditionals having a rank, or a weight, or none). Although the proposed approach provides a framework for conditional reasoning in ASP under different preferential interpretations of conditionals, the paper focuses on weighted KBs, as the basis for the conditional extension of ASP. An extended example of commonsense reasoning over a CondASP program is provided in Section 3.

To deal with the verification of general conditionals $\mathbf{T}(A) \rightarrow B$ over an CondASP program, the formalism is complemented with an approach for *combining preferences*, to define a preference relation $\leq_A$ with respect to a *boolean combination* A of the distinguished propositions A_i. This is discussed in Section 4, where it is proved that entailment of conditional implications $\mathbf{T}(A) \rightarrow B$ from a CondASP program satisfies the *KLM properties* of preferential entailment [27]. Section 5 provides an ASP approach

for verifying entailment from a CondASP program, which also provides a *complexity upper-bound* for conditional entailment.

2 A Multi-Preferential Logic with Typicality

In this section we define a two-valued preferential logic with typicality, which generalizes Kraus Lehmann and Magidor preferential semantics [27, 28], by allowing for multiple preference relations (i.e., preferences with respect to multiple aspects), rather that a single preference relation.

Let $\mathscr{L}$ be a propositional logic, whose formulae are built from a set *Prop* of propositional variables using the boolean connectives $\wedge$, $\vee$, $\neg$ and $\rightarrow$, as usual. We assume that $\bot$ (representing falsity) and $\top$ (representing truth) are formulae of $\mathscr{L}$.

We extend $\mathscr{L}$ by introducing a typicality operator as done in [20] and [8] for the propositional calculus. Intuitively, "a sentence of the form $\mathbf{T}(\alpha)$ is understood to refer to the *typical situations in which* α *holds*" [8]. The typicality operator allows for the formulation of *conditional implications* (or *defeasible implications*) of the form $\mathbf{T}(\alpha) \rightarrow \beta$ whose meaning is that "normally, if α then β", or "in the typical situations when α holds, β also holds". They correspond to conditional implications $\alpha \mathrel{|\!\sim} \beta$ of KLM preferential logics [28], but here we consider a preferential semantics which exploits multiple preferences. As in PTL [8], the typicality operator cannot be nested. When α and β do not contain occurrences of the typicality operator, an implication $\alpha \rightarrow \beta$ is called *strict*. We call $\mathscr{L}^{\mathbf{T}}$ the logic obtained by extending $\mathscr{L}$ with a unary typicality operator $\mathbf{T}$. In the logic $\mathscr{L}^{\mathbf{T}}$, *general implications* $\alpha \rightarrow \beta$ are allowed, where α and β may contain occurrences of the typicality operator.

As mentioned above, the interpretation of a typicality formula $\mathbf{T}(\alpha)$ is defined with respect to a *multi-preferential interpretation*. The KLM preferential semantics [27, 28, 33] exploits a set of worlds $\mathscr{W}$, with their valuation and a preference relation $<$ among worlds (where $w < w'$ means that world w is more normal than world w'). A conditional $A \mathrel{|\!\sim} B$ is satisfied in a KLM preferential interpretation, if B holds in all the most normal worlds satisfying A, i.e., in all $<$-minimal worlds satisfying A.

Here, instead, we consider a multi-preferential semantics, where preference relations are associated with distinguished propositional variables $A_1, \ldots, A_n \in$ *Prop* (called *distinguished propositions* in the following). The idea is that how much a situation (a world) is normal (or less atypical) with respect to another one depends on the aspects considered for comparison. The semantics exploits a set of preference relations $<_{A_i}$, each associated to a distinguished proposition A_i, where $w <_{A_i} w'$ means that world w is less atypical than world w' concerning aspect A_i. For instance, referring to the example in the introduction, a world w can be regarded as capturing a more plausible situation describing a student than world w' (i.e., $w <_{student} w'$), but a less plausible situation than w' describing an employee (i.e., $w' <_{employee} w$).

Multi-preferential semantics have been previously considered for defining refinements of the rational closure construction [19], for defining preferences with respect to different modules of a conditional knowledge base [24], for defining a concept-wise semantics in ranked defeasible $\mathscr{EL}^{\bot}$ knowledge bases [23], and weighted conditionals in *many-valued* DLs with typicality [1]. Here, we deal with a *two-valued* multi-preferential logic with typicality.

Given a finite set of *distinguished propositions* $A_1, \ldots, A_n$, we let $\leq_{A_i} \subseteq W \times W$ be the *preorder* associated with A_i ($w \leq_{A_i} w'$ means that w is not less plausible than w'). A strict partial order $<_{A_i}$ and an equivalence relation $\sim_{A_i}$ can be defined as usual from the preorder $\leq_{A_i}$ by letting $w <'_{A_i}$ iff $w \leq_{A_i} w'$ and $w' \not\leq_{A_i} w$, and letting $w \sim_{A_i} w'$ iff $w \leq_{A_i} w'$ and $w' \leq_{A_i} w$.

2.1 Multi-preferential semantics

In the following, we limit our consideration to finite KBs, and restrict our attention to preferential interpretations with a finite set of preference relations $<_{A_i}$, one for each distinguished proposition A_i. For the moment, we assume that, in any typicality formula $\mathbf{T}(A)$, A is a distinguished proposition. In Section 4 we will lift this restriction.

Definition 1 *A* (multi-)preferential interpretation *is a triple* $\mathscr{M} = \langle \mathscr{W}, \{<_{A_i}\}, v\rangle$ *where:*

- $\mathscr{W}$ *is a non-empty set of worlds;*
- *each* $<_{A_i} \subseteq \mathscr{W} \times \mathscr{W}$ *is an irreflexive and transitive relation on* $\mathscr{W}$*;*
- $v : \mathscr{W} \longrightarrow 2^{Prop}$ *is a valuation function, assigning to each world* w *a set of propositional variables in Prop, the variables which are true in* w*.*

A *ranked interpretation* is a (multi-)preferential interpretation $\mathscr{M} = \langle \mathscr{W}, \{<_{A_i}\}, v\rangle$ for which all preference relations $<_{A_i}$ are *modular*, that is: for all x, y, z, if $x <_{A_i} y$ then $x <_{A_i} z$ or $z <_{A_i} y$ (the preorder $\leq_{A_i}$ is total). A relation $<_{A_i}$ is *well-founded* if it does not allow for infinitely descending chains of worlds $w_0, w_1, w_2, \ldots$ with $w_1 <_{A_i} w_0$, $w_2 <_{A_i} w_1$, ...

The valuation v is inductively extended to all formulae of $\mathscr{L}^{\mathbf{T}}$:

$\mathscr{M}, w \models \top$ $\qquad\qquad$ $\mathscr{M}, w \not\models \bot$

$\mathscr{M}, w \models p$ iff $p \in v(w)$, for all $p \in Prop$

$\mathscr{M}, w \models A \wedge B$ iff $\mathscr{M}, w \models A$ and $\mathscr{M}, w \models B$

$\mathscr{M}, w \models A \vee B$ iff $\mathscr{M}, w \models A$ or $\mathscr{M}, w \models B$

$\mathscr{M}, w \models \neg A$ iff $\mathscr{M}, w \not\models A$

$\mathscr{M}, w \models A \rightarrow B$ iff $\mathscr{M}, w \models A$ implies $\mathscr{M}, w \models B$

$\mathscr{M}, w \models \mathbf{T}(A_i)$ iff $\mathscr{M}, w \models A_i$ and $\nexists w' \in \mathscr{W}$ s.t. $w' <_{A_i} w$ and $\mathscr{M}, w' \models A_i$.

Whether $\mathbf{T}(A_i)$ is satisfied at a world w also depends on the other worlds of the interpretation $\mathscr{M}$.

Let $[[A]]^{\mathscr{M}}$ be the set of all the worlds in $\mathscr{M}$ satisfying a formula A (i.e., $[[A]]^{\mathscr{M}} = \{w \in \mathscr{W} : \mathscr{M}, w \models A\}$) and let $Min_{<_A}(\mathscr{S})$ be the set of $<_A$-minimal worlds in $\mathscr{S}$, for any set of worlds $\mathscr{S} \subseteq \mathscr{W}$, and strict partial order $<_A$, that is:

$$Min_{<_A}(\mathscr{S}) = \{w \in \mathscr{S} \mid \text{ there is no } w' \in \mathscr{S}, \text{ such that } w' <_A w\}$$

Note that, when $\mathscr{S} \neq \emptyset$ and the preference relation $<_A$ is well-founded, the set $Min_{<_A}(\mathscr{S})$ is non-empty. For a *well-founded preference relation* $<_A$, one can reformulate the semantic condition for the typicality operator as follows:

$$\mathscr{M}, w \models \mathbf{T}(A_i) \text{ iff } w \in Min_{<_{A_i}}([[A_i]]^{\mathscr{M}}).$$

We say that a formula A *is satisfiable* in the multi-preferential semantics if there exist a multi-preferential interpretation $\mathscr{M} = \langle \mathscr{W}, \{<_{A_i}\}, v\rangle$ and a world $w \in \mathscr{W}$ such that $\mathscr{M}, w \models A$. A formula A *is valid in an interpretation* $\mathscr{M}$ (written $\mathscr{M} \models A$) if, for all worlds $w \in \mathscr{W}$, $\mathscr{M}, w \models A$. A formula A is *valid* in the multi-preferential semantics (simply, A *is valid*) if A is valid in any multi-preferential interpretation $\mathscr{M}$. Restricting our consideration to ranked interpretations, leads to the notions of satisfiability and validity of a formula in the *ranked (or rational) multi-preferential semantics*.

As mentioned above, when a defeasible implication has the form $\mathbf{T}(A) \rightarrow B$, with B in $\mathscr{L}$, it stands for a conditional $A \mathrel{|\!\sim} B$ in KLM logics [27]. Note that, if we assume that all preference relations are well-founded, a defeasible implication $\mathbf{T}(A) \rightarrow B$ is valid in a preferential interpretation $\mathscr{M}$ (i.e., $\mathscr{M} \models$

$\mathbf{T}(A) \to B$) *iff* for all worlds $w \in \mathscr{W}$, $w \in Min_{<_A}([[A]]^{\mathscr{M}})$ implies $w \in [[B]]^{\mathscr{M}}$, i.e., *iff* $Min_{<_A}([[A]]^{\mathscr{M}}) \subseteq [[B]]^{\mathscr{M}}$ holds. In this case, if we assume that all the preference relations $<_A$ coincide with a single well-founded preference relation $<$, then a multi-preferential interpretation $\mathscr{M}$ becomes a preferential interpretation as in KLM semantics:

$$\mathscr{M} \models \mathbf{T}(A) \to B \textit{ iff } Min_{<}([[A]]^{\mathscr{M}}) \subseteq [[B]]^{\mathscr{M}},$$

i.e., $\mathscr{M} \models \mathbf{T}(A) \to B$ holds when the conditional formula $A \mathrel{|\!\sim} B$ is satisfied in the preferential model $\mathscr{M}$ according to the preferential KLM semantics [27] (resp., to the rational KLM semantics, when $<$ is assumed to be modular). The multi-preferential semantics is indeed a generalization of the KLM preferential semantics.

Let a *knowledge base* K be a set of (strict or defeasible) implications. A *preferential model of* K is a multi-preferential interpretation $\mathscr{M}$ such that $\mathscr{M} \models A \to B$, for all implications $A \to B$ in K. Given a knowledge base K, we say that an implication $A \to B$ *is preferentially entailed from* K if $\mathscr{M} \models A \to B$ holds, for all preferential models $\mathscr{M}$ of K. We say that $A \to B$ *is rationally entailed from* K if $\mathscr{M} \models A \to B$ holds, for all ranked models $\mathscr{M}$ of K.

It is well known that preferential entailment and rational entailment are weak. As with the rational closure [28] and the lexicographic closure [29] for KLM conditionals, also in the multi-preferential case one can strengthen entailment by restricting to specific preferential models, based on some *closure constructions*, which allow to define the preference relations $<_{A_i}$ from a knowledge base K, also exploiting the ranks and weights of conditional implications, when available. Some examples of closure constructions for the multi-preferential case have been considered, e.g., for variants of the rational closure [19] and of the lexicographic closure [24], and for ranked or weighted defeasible DLs with typicality [23, 1].

In the next section, we define an approach for combining an ASP program with a conditional knowledge base, and develop a construction based on the weighted conditional KB for reasoning on the answer sets of the ASP programs.

3 A conditional extension of Answer Set Programming

In this section, we aim at combining a defeasible knowledge base with an ASP Program, to exploit conditionals to define preferences between the Answer Sets of the ASP program. Based on such preferences, conditional properties of the program can be validated over a preferential interpretation of the program, based on the semantics of the conditional logic with typicality introduced in the previous section.

More precisely, a *Conditional ASP (CondASP) program* is a pair (Π, K), where Π is an ASP program and K is a defeasible knowledge base. Both Π and K rely on the same set of propositional variables *Prop*. In this paper we specifically consider weighted knowledge bases, and assume that K is a weighted KB, i.e., a set of conditional implications, with an associated integer weight.

For simplicity, let $A_1, \ldots, A_n$ be a set of distinguished propositional variables. A *weighted defeasible knowledge base* K over the distinguished propositions $A_1, \ldots, A_n$ is a set of *defeasible implications* of the form $(\mathbf{T}(A_i) \sqsubseteq B_{i,j}, h_{i,j})$, where A_i is a distinguished proposition, $B_{i,j} \in \mathscr{L}$ and the weight $h_{i,j}$ is an integer number. The weight is intended to represent the *plausibility* (or *implausibility*) of the associated defeasible implication, where negative weights represent penalties.

Let us consider the following example program, which is inspired by the cinema scenario in [12]. On Saturday evening, two married persons, as Bob and Mary, can either go to the cinema, go for a pizza or stay at home. But they cannot both go out if they have children and a trusted babysitter is not available.

$$1\{go_to_cinema(X); go_for_pizza(X); stay_at_home(X)\}1 \text{ :- } person(X).$$

$1\{stay_at_home(X); stay_at_home(Y)\} :- married(X,Y), have_children(X,Y),$
$\quad not\ available_babysitter_for(X,Y).$
$available_babysitter_for(X,Y) :- available_babysitter_for(X,Y,Z).$
$available_babysitter_for(X,Y,Z) :- married(X,Y), babysitter(Z),$
$\quad not\ busy(Z), 1\{trust(X,Z); trust(Y,Z)\}.$
$:- go_to_cinema(Z), available_babysitter_for(X,Y,Z).$
$:- go_for_pizza(Z), available_babysitter_for(X,Y,Z).$
$\{housekeeping(X)\} :- stay_at_home(X).$
$busy(X) :- go_to_cinema(X).$ $\quad busy(X) :- go_for_pizza(X).$
$married(bob,mary).$ $\quad have_children(bob,mary).$
$babysitter(ada).$ $\quad trust(mary,ada).$
$person(X) :- babysitter(X).$
$person(X) :- married(X,Y).$ $\quad person(Y) :- married(X,Y).$
$:- trust(mary,Z), Z! = ada.$
$:- go_for_pizza(mary), not\ go_for_pizza(bob).$
$:- go_for_pizza(bob), not\ go_for_pizza(mary).$
$:- happy_Sat(bob), housekeeping(bob).$
$\{happy_Sat(X)\} :- person(X).$

We have assumed that Mary and Bob would only have a pizza together, but they can go to the cinema independently. Let us assume that Mary, Bob and Ada have their preferences for Saturday evening, which are expressed by the following weighted knowledge base.

$\mathbf{T}(happy_Sat(mary)) \rightarrow go_to_cinema(mary), +40$
$\mathbf{T}(happy_Sat(mary)) \rightarrow stay_at_home(mary), -10$
$\mathbf{T}(happy_Sat(mary)) \rightarrow go_for_pizza(mary), +20$
$\mathbf{T}(happy_Sat(bob)) \rightarrow go_to_cinema(bob), +10$
$\mathbf{T}(happy_Sat(bob)) \rightarrow stay_at_home(bob), -30$
$\mathbf{T}(happy_Sat(bob)) \rightarrow go_for_pizza(bob), +40$
$\mathbf{T}(happy_Sat(ada)) \rightarrow go_to_cinema(ada), +30$
$\mathbf{T}(happy_Sat(ada)) \rightarrow go_to_cinema(mary), +30$

While Mary has a preference for going to the cinema, Bob is more fond of pizza. Both of them dislike staying at home on the Saturday evening (and, from the ASP program above, we know that Bob would not have a happy Saturday doing housekeeping). Concerning Ada's preferences, she likes going to the cinema. However, as she is Mary's friend, she is equally happy when Mary can go to the cinema on Saturday.

Which are the typical situations in which Mary is happy, or Mary and Bob are happy on Saturday evening? And what does also hold in such situations? We aim at using the conditional logic to answer such questions. For instance, considering the typical situations in which Bob and Mary are happy on Saturday, are these situations where they go for pizza, i.e., $\mathbf{T}(happy_Sat(mary) \wedge happy_Sat(bob)) \rightarrow go_for_pizza(mary) \wedge go_for_pizza(bob)$? Further, are these typical situations in which Mary is happy on Saturday, i.e., $\mathbf{T}(happy_Sat(mary) \wedge happy_Sat(bob)) \rightarrow \mathbf{T}(happy_Sat(mary))$?

In the following we describe how a weighted knowledge base can be used to define preferences over the answer sets of an ASP program (see e.g. [17] for the definition of the semantics of ASP programs including constraints and choice rules used in the example above).

3.1 Weighted knowledge bases and preferences over Answer Sets

Given a weighted CondASP program (Π, K), if Π has answer sets, we can construct a multi-preferential interpretation $\mathscr{M}_K^\Pi$ such that the set of worlds $\mathscr{W}$ is the set of the answer sets of the program Π, the valuation function maps each answer set S into the corresponding propositional interpretation, and the preference relations $<_{A_i}$ on the answer sets of Π are defined from the set of weighted conditionals in K having the form $(\mathbf{T}(A_i) \rightarrow B_{i,j}\ , h_{i,j})$.

As an answer set S of an ASP program Π is a set of propositional variables in *Prop*, it can be regarded as a propositional interpretation (in which the variables in S are true, and the others are false), and the satisfiability in S of any boolean formula is defined as usual. The preference relation $<_{A_i}$ associated with the distinguished proposition A_i is determined by computing, for each answer set $S \in \mathscr{W}$, the weight $W_{A_i}(S)$ of S with respect to proposition A_i:

$$W_{A_i}(S) = \sum_{j: S \models B_{i,j}} h_{i,j} \tag{1}$$

Informally, the weight $W_{A_i}(S)$ of S wrt A_i is the sum of the weights of all defeasible implications $\mathbf{T}(A_i) \rightarrow B_{i,j}$ for A_i, such that $B_{i,j}$ is satisfied by S. The more plausible are the properties satisfied by S in the typical situations for A_i, the higher is the weight of S with respect to A_i. If no defeasible implication $(\mathbf{T}(A_i) \rightarrow B_{i,j}\ , h_{i,j})$ is in K for a distinguished proposition A_i, then $W_{A_i}(S) = 0$ holds for all answer sets S. The preorder relation $\leq_{A_i}$ associated to proposition A_i is defined as follows:

$$S_1 \leq_{A_i} S_2 \quad \text{iff} \quad W_{A_i}(S_1) \geq W_{A_i}(S_2). \tag{2}$$

A CondASP program (Π, K) is *consistent* if Π has at least an answer set. Given a consistent CondASP program (Π, K), we define a unique multi-preferential model of (Π, K), as follows.

Definition 2 *The* multi-preferential model *of a consistent CondASP program* (Π, K) *is a preferential interpretation* $\mathscr{M}_K^\Pi = \langle \mathscr{W}, \{<_{A_i}\}, v\rangle$ *s.t.:*

- $\mathscr{W}$ *is the set of all the answer sets of* Π*;*
- *for all* $S \in \mathscr{W}$, $v(S) = S$*;*
- *for all distinguished propositions* A_i, $<_{A_i}$ *is the strict relation induced by the preorder* $\leq_{A_i}$*, that is: for all* $S_1, S_2 \in \mathscr{W}$, $S_1 <_{A_i} S_2$ *iff* $W_{A_i}(S_1) > W_{A_i}(S_2)$.

An answer set S_1 is *preferred to an answer set* S_2 with respect to A_i (written $S_1 <_{A_i} S_2$), if the weight of S_1 with respect to A_i is greater than the weight of S_2 with respect to A_i; and vice-versa.

We regard $\mathscr{M}_K^\Pi$ as the *canonical model* of (Π, K).[1]. To clarify the definition of preferences, let us consider again the example about students and employees, from the introduction.

Example 1 *Consider a CondASP program* (Π, K)*, such that, for instance:*

$S_1 = \{student, employee, adult, has_SSN, young, has_classes\}$

$S_2 = \{student, employee, adult, has_SSN, has_boss\}$

are both answer sets of Π*, and K consists of the defeasible properties* $(d_1) - (d_6)$ *from the introduction. The weight of* S_1 *with respect to proposition student is* $W_{student}(S_1) = 90 + 80 = 170$*, while the weight of* S_2 *with respect to proposition student is* $W_{student}(S_2) = -80$*. Hence,* $S_1 <_{student} S_2$*. On the other hand,* $W_{employee}(S_1) = -120$*, and* $W_{employee}(S_2) = 100$*. Hence,* $S_2 <_{employee} S_1$.

Given a consistent CondASP program (Π, K), and an implication $A \rightarrow B$, we say that $A \rightarrow B$ *is entailed from the program* (Π, K) (written $(\Pi, K) \models A \rightarrow B$), if $\mathscr{M}_K^\Pi \models A \rightarrow B$ holds.

[1] Restricting to preferential models over a set of propositional interpretations is a standard approach in KLM logics, e.g., in the semantics of rational closure and of lexicographic closure of a KB [28, 29].

3.2 Other preference definitions: Preferences based on the ranks of conditionals

When the defeasible knowledge base is a set of conditional implications $\mathbf{T}(A) \rightarrow B$, with $A, B \in \mathscr{L}$, computing the Rational Closure or the Lexicographic closure of the knowledge base are viable options to define a single preorder relation $\leq$ on the answer sets of the program Π. In this case, the ranks can be computed according to the rational closure construction, e.g., using Datalog with stratified negation, as in the Datalog encoding of the rational closure of the lightweight description logic $\mathscr{SROEL}$ [25].

In the previous section, we have seen that the *(user defined) weights* can be exploited to define the different preorder relations $\leq_{A_i}$ for the distinguished propositions A_i. Another option is that *(user defined) ranks* $0, 1, 2, \ldots$ are associated to the conditional implications in the knowledge base K, and that conditionals with higher ranks are considered to be more important than conditionals with a lower ranks. As for weights, one can use the ranks associated to the conditional implications $\mathbf{T}(A_i) \rightarrow B_{i,j}$ for a distinguished proposition A_i, to define the preorder relation $\leq_{A_i}$. This way to define the preferences relations has been exploited, for instance, in a defeasible description logic with typicality [23], in which preferences are associated to distinguished concepts C_i, but the idea of having ranked knowledge bases with user-defined ranks was previously explored in Brewka's framework for qualitative preferences [11, 12], in which basic preference relations $\geq_K$ are associated with different *ranked knowledge bases* K.

In the general case, an interesting option can be allowing different ways of specifying preferences for different distinguished propositions A_i *in the same knowledge base*. As an example, let us define the typical properties of *horses*, by a set of conditional implications with a rank:

(d_7) $\mathbf{T}(horse) \rightarrow has_Saddle$, 0 $\quad$ (d_9) $\mathbf{T}(horse) \rightarrow run_fast$, 1

(d_8) $\mathbf{T}(horse) \rightarrow has_Long_Mane$, 0 $\quad$ (d_{10}) $\mathbf{T}(horse) \rightarrow has_tail$, 1

where the properties of having a tail and running fast are regarded as being more important (for a horse) than the properties of having a saddle and a long mane.

Adopting the definition in [23], one can define the preference relation $\leq_{A_i}$ associated with distinguished proposition A_i,considering the ranks of the defeasible implications of the form $\mathbf{T}(A_i) \rightarrow B_{j,i}$ in K, by letting $\mathscr{T}^l_{A_i}(S)$ be the set of typicality inclusions for A_i with rank l, being satisfied by S, that is:

$$\mathscr{T}^l_{A_i}(S) = \{\mathbf{T}(A_i) \rightarrow B_{j,i} \mid (\mathbf{T}(A_i) \rightarrow B_{j,i}, l) \in K \text{ and } S \not\models A_i \text{ or } S \models B_{j,i}\}.$$

Given two answer sets S_1 and S_2 of the program Π, we let

$$S_1 \leq_{A_i} S_2 \quad \text{iff} \quad \text{either } |\mathscr{T}^l_{A_i}(S_1)| = |\mathscr{T}^l_{A_i}(S_2)|, \text{ for all } l,$$
$$\text{or } \exists l \text{ such that } |\mathscr{T}^l_{A_i}(S_1)| > |\mathscr{T}^l_{A_i}(S_2)| \text{ and, } \forall h > l,\ |\mathscr{T}^h_{A_i}(x_1)| = |\mathscr{T}^h_{A_i}(S_2)|.$$

Informally, the preference relation $\leq_{A_i}$ gives higher preference to answer sets violating a smaller number of conditional implications with higher rank for A_i. It corresponds to the strategy # in Brewka's framework for qualitative preferences [12], which defines a total preorder. We refer to [23] for an encoding in *asprin* [13] of the preferences $\leq_{A_i}$, used therein for reasoning from a ranked $\mathscr{SROEL}$ knowledge base.

In a general framework, the ranked conditionals $(d_7) - (d_{10})$ specifying the preorder $\leq_{horse}$, can belong to the same knowledge base K as the weighted conditionals $(d_1) - (d_6)$ specifying the preference relations $\leq_{student}$ and $\leq_{employeed}$. However, in the following, we will restrict our attention to knowledge bases only containing weighted conditionals, and develop an ASP encoding of entailment from a CondASP program (Π, K), with K a weighted KB.

Let us first consider the issue of preference combination for a CondASP program (Π, K), with K a weighted conditional KB.

4 Combining preferences from weighted knowledge bases

In previous approaches to multi preferential semantics in the two-valued case, e.g., [19, 24, 23], a global preference relation $<$ is defined from the base preferences $<_{A_i}$; this allows to provide a meaning of a defeasible implication $\mathbf{T}(A) \rightarrow B$, for any formula A, by letting:

$$\mathscr{M} \models \mathbf{T}(A) \rightarrow B \textit{ iff } Min_{<}([[A]]^{\mathscr{M}}) \subseteq [[B]]^{\mathscr{M}},$$

However, choosing a single global preference relation $<$ may be restrictive, as one may want to exploit different minimization criteria and allow for different user-defined preference relations. In this paper, rather than defining a single global preference relation $<$, we allow for the definition of different preference relations, associated with complex formulae obtained by combining the distinguished propositions $A_1,\ldots,A_n$. For instance, referring to the example in Section 3, we aim at considering the most plausible situations in which both Mary and Bob are happy on Saturday.

In Brewka's framework for preference combination [12], a logical preference description language is introduced for combining basic preference descriptions d_1 and d_2 (defining preorders on models) into complex preference descriptions $d_1 \wedge d_2$, $d_1 \vee d_2$, $\neg d_1$ and $d_1 > d_2$ (where $d_1 > d_2$ is intended to express the priority of preference d_1 over preference d_2). However, as observed by Brewka [12], the preference description $\neg(d_1 \vee d_2)$ is different from $\neg d_1 \wedge \neg d_2$. This would lead to break some wanted KLM properties of a nonmonotonic consequence relation [27], such as *Left Logical Equivalence* (see below).

In the following, for weighted KBs, we propose an alternative approach for preference combination, which exploits the weights $W_{A_i}(S)$, introduced to define the preference relations $<_{A_i}$ in $\mathscr{M}_K^{\Pi}$. We first extend, in Definition 1, the semantic condition for the typicality operator to all typicality formulae $\mathbf{T}(A)$ such that A is a boolean combination of $A_1,\ldots,A_n$ (i.e., A is obtained by combining the distinguished propositions A_i using $\wedge$, $\vee$ and $\neg$, while $\bot$ and $\top$ are not allowed to occur in A). As the set $\mathscr{W}$ of answer sets of Π is finite, we can restrict to well-founded preference relations, and let:

$$\mathscr{M}_K^{\Pi}, S \models \mathbf{T}(A) \text{ iff } S \in Min_{<_A}([[A]]^{\mathscr{M}_K^{\Pi}}).$$

For evaluating a typicality formula $\mathbf{T}(A)$ at a world, we need to define a preference relation $\leq_A$ associated to a formula A. For instance, for determining the most normal situations in which both Bob and Mary are happy on Saturday ($\mathbf{T}(happy_Sat(bob) \wedge happy_Sat(mary))$ we need to define the preference relation $<_{happy_Sat(bob) \wedge happy_Sat(mary)}$.

To define the preorder relation $\leq_A$ associated to a complex formula A (a boolean combination of the A_i's) we inductively extend to complex formulae the notion of the weight of an answer set with respect to a formula. Given (Π, K), let Max and Min be, resp., the maximum and the minimum value of the weight $W_{A_i}(S)$, for each distinguished proposition A_i in K and answer set S. We let:

$$W_{A_1 \wedge A_2}(S) = min(W_{A_1}(S), W_{A_2}(S)) \qquad W_{A_1 \vee A_2}(S) = max(W_{A_1}(S), W_{A_2}(S))$$
$$W_{\neg A_i}(S) = Max - W_{A_i}(S) + Min$$

Proposition 1 *The following properties hold:*

$$W_{\neg\neg A}(S) = W_A(S)$$
$$W_{\neg(A \wedge B)}(S) = W_{\neg A \vee \neg B}(S) \qquad W_{\neg(A \vee B)}(S) = W_{\neg A \wedge \neg B}(S)$$
$$W_{A \vee B}(S) = W_{B \vee A}(S) \qquad W_{A \wedge B}(S) = W_{B \wedge A}(S)$$
$$W_{A \vee A}(S) = W_A(S) \qquad W_{A \wedge A}(S) = W_A(S)$$
$$W_{(A \vee B) \vee C}(S) = W_{A \vee (B \vee C)}(S) \qquad W_{(A \wedge B) \wedge C}(S) = W_{A \wedge (B \wedge C)}(S)$$

As a consequence, two equivalent formulae have the same weight in an answer set S:

Lemma 1 *Let A and B be boolean combinations of the distinguished propositions $A_1,\ldots,A_n$. If A and B are equivalent in propositional logic, then $W_A(S) = W_B(S)$, for all answer sets S.*

The preference relation associated with a complex formula can be defined by extending condition (2) to any formula A which is a boolean combination of distinguished propositions. For all $S_1, S_2 \in \mathscr{W}$, we let:

$$S_1 \leq_A S_2 \quad \text{iff} \quad W_A(S_1) \geq W_A(S_2).$$

The preference relation $\leq_A$ is a total preorder. As a direct consequence of Lemma 1, the following proposition holds:

Proposition 2 *Let A and B be boolean combinations of the distinguished propositions $A_1,\ldots,A_n$. If A and B are equivalent in propositional logic, then, for all answer sets $S_1, S_2 \in \mathscr{W}$, $S_1 \leq_A S_2$ iff $S_1 \leq_B S_2$.*

Kraus, Lehmann and Magidor [27] studied the properties of several families of nonmonotonic *consequence relations*, i.e., binary relations $\mathrel{|\!\sim}$ on $\mathscr{L}$, corresponding to well-behaved sets of conditional assertions. In particular, they introduced a notion of *preferential consequence relation* as a consequence relation satisfying the following properties (also called KLM postulates) that we reformulate using the typicality operator[2]:

(*Reflexivity*) $\mathbf{T}(A) \rightarrow A$

(*Right Weakening*) If $\models B \rightarrow C$ and $\mathbf{T}(A) \rightarrow B$ then $\mathbf{T}(A) \rightarrow C$

(*Left Logical Equivalence*) If $\models A \equiv B$ and $\mathbf{T}(A) \rightarrow C$ then $\mathbf{T}(B) \rightarrow C$

(*And*) If $\mathbf{T}(A) \rightarrow B$ and $\mathbf{T}(A) \rightarrow C$ then $\mathbf{T}(A) \rightarrow B \wedge C$

(*Or*) If $\mathbf{T}(A) \rightarrow C$ and $\mathbf{T}(B) \rightarrow C$ then $\mathbf{T}(A \vee B) \rightarrow C$

(*Cautious Monotonicity*) If $\mathbf{T}(A) \rightarrow B$ and $\mathbf{T}(A) \rightarrow C$ then $\mathbf{T}(A \wedge B) \rightarrow C$

We prove that the entailment from a CondASP program (Π, K) is a preferential consequence relation. Let us remember that $\mathbf{T}(A) \rightarrow B$ is entailed from the program (Π, K), if $\mathscr{M}_K^\Pi \models \mathbf{T}(A) \rightarrow B$ holds.

Proposition 3 *Given a consistent CondASP program (Π, K), entailment from (Π, K) satisfies the KLM postulates above.*

For instance, to prove that the property *(Or)* is satisfied, one has to prove that, assuming that $\mathscr{M}_K^\Pi \models \mathbf{T}(A) \rightarrow C$ and $\mathscr{M}_K^\Pi \models \mathbf{T}(B) \rightarrow C$ hold, then $\mathscr{M}_K^\Pi \models \mathbf{T}(A \vee B) \rightarrow C$ also holds. The detailed proof of the proposition can be found in the Appendix of [2].

One observation is that the property of Rational Monotonicity:

(*RM*) If $\mathbf{T}(A) \rightarrow C$ and $\mathbf{T}(A) \not\rightarrow \neg B$ then $\mathbf{T}(A \wedge B) \rightarrow C$

is not satisfied by the entailment from a CondASP program (Π, K). In general, it is not the case that, if $\mathscr{M}_K^\Pi \models \mathbf{T}(A) \rightarrow C$ and $\mathscr{M}_K^\Pi \not\models \mathbf{T}(A) \rightarrow \neg B$, it follows that $\mathscr{M}_K^\Pi \models \mathbf{T}(A \wedge B) \rightarrow C$.

As an example, suppose that from a conditional program (Π, K), we can conclude that typical birds fly and that there are some typical birds which are *black&white* (e.g., the warbler). Should we require that the most typical birds which are *black&white* do fly? For instance, the most typical *black&white* birds might be the penguins, and we do not want to conclude that they fly. In this case, it looks reasonable that (RM) does not hold (a detailed counterexample is in the appendix in [2]).

[2] In the KLM postulates below, $\models A \equiv B$ means that A and B are equivalent formulae in propositional logic, and $\models B \rightarrow C$ means that the implication $B \rightarrow C$ is valid in propositional logic

5 Verifying conditional implications in ASP

We describe how ASP can be used to verify entailment of an implication from a program (Π, K).

In order to use meta-programming, the program Π is transformed replacing all atoms A with $holds(A)$. A weighted defeasible implication $(\mathbf{T}(A_i) \sqsubseteq B_{i,j}, h_{i,j})$ in K is represented with a fact $wdi(A_i, B_{i,j}, h_{i,j})$. Then, the weight with respect to an atomic formula A can be computed using $\#sum\{W1, B : wdi(A, B, W1), holds(B)\}$.

For the case of conditional implications $\mathbf{T}(A) \rightarrow B$, where A, B are boolean formulae, a solution can be adapted from [3] (where an ASP encoding of entailment in a defeasible description logic with typicality was considered in the many-valued case). The implication $\mathbf{T}(A) \rightarrow B$ to be verified is represented as $query(typ(A), B)$, using, in A and B, symbols $and/2, or/2, neg/1$ for representing the structure of the formulae. The following constraint (with additional rules to extend *holds* to non-atomic formulae):

$$:\!- \; query(typ(A), B), not\ holds(A).$$

is used to select answer sets of Π in which A holds. If there are none, the implication is trivially true. The following weak constraint is used to prefer (at priority level 2) answer sets of Π where the weight of A is maximum, i.e., the answer sets in $Min_{<_A}([[A]]^{\mathscr{M}_K^\Pi})$:

$$:\!\sim \; query(typ(A), B), weight(A, W).\ [-W@2]$$

The following rules and weak constraint:

$$rhs :\!- \; query(_, B), holds(B).$$

$$counterexample :\!- \; not\ rhs.$$

$$:\!\sim \; counterexample.\ [-1@1]$$

are used to prefer, at level 1, the presence of *counterexample*: in this way, if there is an answer set of Π which falsifies the implication $\mathbf{T}(A) \rightarrow B$, we get *counterexample* in all answer sets that are optimal with respect to the weak constraints; in none, otherwise. Then, a single ASP solver call computing a single optimal answer set is enough to establish entailment of the implication.

An example formula that is entailed for the conditional ASP program in section 3 is:

$$\mathbf{T}(happy_Sat(mary)) \rightarrow go_to_cinema(mary)$$

General implications $A \rightarrow B$ with $\mathbf{T}$ occurring in A and B can be dealt with as follows. A solver call is performed for each occurrence $\mathbf{T}(C)$ of $\mathbf{T}$ in A, B, to establish, using a weak constraint, the maximum value of $W_C(S)$ across the answer sets S of Π. Then, a further solver call is performed to see if there is an answer set of Π which is a counterexample for $A \rightarrow B$, evaluating the truth of formulae $\mathbf{T}(C)$ with:

$$holds(typ(C)) :\!- \; holds(C), maxw(C, MW), weight(C, MW).$$

where *maxw* is used to represent the maximum weight computed before. This encoding provides a P^{NP} upper bound to the complexity of entailment [14].

Some implications that result to be entailed by the conditional ASP program in Section 3 are:

$$\mathbf{T}(happy_Sat(mary)) \rightarrow \neg\mathbf{T}(happy_Sat(bob))$$

$$\mathbf{T}(happy_Sat(bob)) \rightarrow \neg\mathbf{T}(happy_Sat(mary))$$

$$\mathbf{T}(happy_Sat(mary) \wedge happy_Sat(bob)) \rightarrow \neg\mathbf{T}(happy_Sat(mary))$$

$$\mathbf{T}(happy_Sat(mary) \wedge happy_Sat(bob))) \rightarrow \mathbf{T}(happy_Sat(bob))$$

since Mary's and Bob's top preferences are different, and the case that maximizes weight for $happy_Sat(mary) \wedge happy_Sat(bob)$ is the one where they both go for pizza, which is Bob's top preference but not Mary's.

The two algorithms described above are implemented in dedicated reasoners based on *ASP Chef* [4], a web-based platform for answer set programming. ASP Chef allows users to compose and execute declarative recipes, combining ASP programs with additional logic-based components, such as conditional knowledge bases. The first algorithm, for queries of the form $\mathbf{T}(A) \rightarrow B$, is available online at *https://asp-chef.alviano.net/s/ICLP2025/conditional-asp-simple*. The second one handles the general case and is accessible at *https://asp-chef.alviano.net/s/ICLP2025/conditional-asp*. In both implementations, users can specify a *query*, a *logic program*, and a *weighted conditional knowledge base* through an intuitive web interface. The system elaborates the inputs by translating them into an ASP-based reasoning process, and computes the result directly in the browser. This makes the tool accessible without requiring any software installation, thus supporting both experimentation and education in conditional reasoning with ASP.

6 Conclusions

In this paper we have developed a framework for conditional reasoning from a Conditional ASP program, combining an ASP program Π with a set K of (weighted or ranked) conditionals over the same set of propositional variables. The framework allows for conditional reasoning over the answer sets of the program Π, by verifying implications including the typicality operator. We have introduced a two-valued multi-preferential semantics generalizing the KLM semantics [27] and, for weighted knowledge bases, we have developed an approach for combining preferences. The paper also provides an ASP approach for verifying entailment from a CondASP program, and corresponding reasoners in ASP Chef.

The idea of associating weights/ranks to the properties of concepts in weighted KBs, as a measure of their saliency, was inspired by Brewka's framework for basic preference descriptions [11, 12], and by Lehmann's lexicographic closure [29]. Among the recent work on preference combination in ASP, let us mention the algebraic framework for preference combination in multi-relational contextual hierarchies proposed by Bozzato et al. [10], and the hybrid approach for combining an ASP program with a conditional knowledge base, proposed by Wilhelm et al. [37] for prioritizing answer sets based on conditional expert knowledge: c-representations [26] are used for ranking the feasible warehouse layouts generated by an ASP program in the logistic domain.

Our aim in this paper was to develop an integrated formalism for conditional reasoning over an extended ASP program, and to provide an ASP encoding of the formalism. Our use of weighted knowledge bases has also some relations with *threshold concepts* in description logics by Baader et al. [6] and with *weighted threshold operators* by Porello et al. [35]. The notion of typicality we have considered is also reminiscent of *prototype theory* [36] and relates to Freund's ordered models for concept representation [16] although, here, we do not have a representation of concepts, but we are considering propositions.

Acknowledgements: This research was partially supported by INDAM-GNCS. Mario Alviano was partially supported by the Italian Ministry of University and Research (MUR) under PRIN project PRODE "Probabilistic declarative process mining", CUP H53D23003420006, under PNRR project FAIR "Future AI Research", CUP H23C22000860006, under PNRR project Tech4You "Technologies for climate change adaptation and quality of life improvement", CUP H23C22000370006, and under PNRR project SERICS "SEcurity and RIghts in the CyberSpace", CUP H73C22000880001; by the Italian Ministry of Health (MSAL) under POS projects CAL.HUB.RIA (CUP H53C22000800006) and RADIOAM-

ICA (CUP H53C22000650006); by the Italian Ministry of Enterprises and Made in Italy under project STROKE 5.0 (CUP B29J23000430005); under PN RIC project ASVIN "Assistente Virtuale Intelligente di Negozio" (CUP B29J24000200005); and by the LAIA lab (part of the SILA labs). Mario Alviano is member of Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

References

[1] M. Alviano, F. Bartoli, M. Botta, R. Esposito, L. Giordano & D. Theseider Dupré (2024): *A preferential interpretation of MultiLayer Perceptrons in a conditional logic with typicality*. Int. Journal of Approximate Reasoning 164. Available at *10.1016/j.ijar.2023.109065*.

[2] M. Alviano, L. Giordano & D. Theseider Dupré (2025): *A framework for Conditional Reasoning in Answer Set Programming*. CoRR abs/2506.03997, doi:*10.48550/ARXIV.2506.03997*.

[3] M. Alviano, L. Giordano & D. Theseider Dupré (2024): *Complexity and Scalability of Defeasible Reasoning in Many-valued Weighted Knowledge Bases with Typicality*. J. Logic and Comput. 34(8), pp. 1469–1499, doi:*10.1093/logcom/exae038*.

[4] M. Alviano & L. Angel Rodriguez Reiners (2024): *ASP Chef: Draw and Expand*. In: Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR, doi:*10.24963/kr.2024/68*.

[5] F. Baader, S. Brandt & C. Lutz (2005): *Pushing the $\mathscr{EL}$ envelope*. In L.P. Kaelbling & A. Saffiotti, editors: Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI 2005), Professional Book Center, Edinburgh, Scotland, UK, pp. 364–369, doi:*10.1007/11551263_4*.

[6] F. Baader, G. Brewka & O. F. Gil (2015): *Adding Threshold Concepts to the Description Logic* EL. In: Frontiers of Combining Systems - 10th International Symposium, Lecture Notes in Computer Science 9322, Springer, pp. 33–48, doi:*10.1007/978-3-319-24246-0_3*.

[7] S. Benferhat, C. Cayrol, D. Dubois, J. Lang & H. Prade (1993): *Inconsistency Management and Prioritized Syntax-Based Entailment*. In: Proc. IJCAI'93, Chambéry,, pp. 640–647.

[8] R. Booth, G. Casini, T. Meyer & I. Varzinczak (2019): *On rational entailment for Propositional Typicality Logic*. Artif. Intell. 277, doi:*10.1016/j.artint.2019.103178*.

[9] R. Booth & J. B. Paris (1998): *A Note on the Rational Closure of Knowledge Bases with Both Positive and Negative Knowledge*. Journal of Logic, Language and Information 7(2), pp. 165–190, doi:*10.1023/A:1008261123028*.

[10] L. Bozzato, T. Eiter & R. Kiesel (2021): *Reasoning on Multirelational Contextual Hierarchies via Answer Set Programming with Algebraic Measures*. Theory Pract. Log. Program. 21(5), pp. 593–609, doi:*10.1017/S1471068421000284*.

[11] G. Brewka (1989): *Preferred Subtheories: An Extended Logical Framework for Default Reasoning*. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence, pp. 1043–1048.

[12] G. Brewka (2004): *A Rank Based Description Language for Qualitative Preferences*. In: 6th Europ. Conf. on Artificial Intelligence, ECAI'2004, Valencia, Spain, August 22-27, 2004, pp. 303–307.

[13] G. Brewka, J. P. Delgrande, J. Romero & T. Schaub (2015): *asprin: Customizing Answer Set Preferences without a Headache*. In: Proc. AAAI 2015, pp. 1467–1474, doi:*10.1609/aaai.v29i1.9398*.

[14] Francesco Buccafurri, Nicola Leone & Pasquale Rullo (1997): *Strong and Weak Constraints in Disjunctive Datalog*. In: LPNMR, Lecture Notes in Computer Science 1265, Springer, pp. 2–17, doi:*10.1016/0004-3702(87)90053-1*.

[15] J. Delgrande (1987): *A First-order Conditional Logic for Prototypical Properties*. Artificial Intelligence 33(1), pp. 105–130, doi:*10.1016/0004-3702(87)90053-1*.

[16] M. Freund (2020): *Ordered models for concept representation*. J. Log. Comput. 30(6), doi:*10.1093/logcom/exaa034*.

[17] M. Gebser, R. Kaminski, B. Kaufmann & T. Schaub (2012): *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, doi:*10.1007/978-3-031-01561-8*.

[18] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In: *Logic Programming, Proceedings of the Fifth International Conference and Symposium*, MIT Press, pp. 1070–1080.

[19] L. Giordano & V. Gliozzi (2021): *A reconstruction of multipreference closure*. Artif. Intell. 290, doi:*10.1016/j.artint.2020.103398*.

[20] L. Giordano, V. Gliozzi, N. Olivetti & G. L. Pozzato (2007): *Preferential Description Logics*. In: *LPAR 2007*, *LNAI* 4790, Springer, Yerevan, Armenia, pp. 257–272, doi:*10.1007/978-3-540-75560-9_20*.

[21] L. Giordano, V. Gliozzi, N. Olivetti & G. L. Pozzato (2009): *ALC+T: a Preferential Extension of Description Logics*. Fundamenta Informaticae 96, pp. 1–32, doi:*10.3233/FI-2009-182*.

[22] L. Giordano & D. Theseider Dupré (2016): *ASP for minimal entailment in a rational extension of SROEL*. *TPLP* 16(5-6), pp. 738–754, doi:*10.1017/S1471068416000399*.

[23] L. Giordano & D. Theseider Dupré (2020): *An ASP approach for reasoning in a concept-aware multipreferential lightweight DL*. *TPLP* 10(5), pp. 751–766, doi:*10.1017/S1471068420000381*.

[24] L. Giordano & D. Theseider Dupré (2020): *A framework for a modular multi-concept lexicographic closure semantics*. In: *Proc. 18th Int. Workshop on Non-Monotonic Reasoning, NMR2020*.

[25] Laura Giordano & Daniele Theseider Dupré (2018): *Defeasible Reasoning in SROEL: from Rational Entailment to Rational Closure*. Fundamenta Informaticae 161(1-2), pp. 135–161, doi:*10.3233/FI-2018-1698*.

[26] G. Kern-Isberner (2001): *Conditionals in Nonmonotonic Reasoning and Belief Revision - Considering Conditionals as Agents*. *LNCS* 2087, Springer, doi:*10.1007/3-540-44600-1*.

[27] S. Kraus, D. Lehmann & M. Magidor (1990): *Nonmonotonic Reasoning, Preferential Models and Cumulative Logics*. Artificial Intelligence 44(1-2), pp. 167–207, doi:*10.1016/0004-3702(90)90101-5*.

[28] D. Lehmann & M. Magidor (1992): *What does a conditional knowledge base entail?* Artificial Intelligence 55(1), pp. 1–60, doi:*10.1016/0004-3702(92)90041-U*.

[29] D. J. Lehmann (1995): *Another Perspective on Default Reasoning*. Ann. Math. Artif. Intell. 15(1), pp. 61–82, doi:*10.1007/BF01535841*.

[30] D. Lewis (1973): *Counterfactuals*. Basil Blackwell Ltd.

[31] David Makinson (1988): *General Theory of Cumulative Inference*. In: *Non-Monotonic Reasoning, 2nd International Workshop*, pp. 1–18, doi:*10.1007/3-540-50701-9_16*.

[32] D. Nute (1980): *Topics in Conditional Logic*. Reidel, Dordrecht, doi:*10.1007/978-94-009-8966-5*.

[33] J. Pearl (1988): *Probabilistic Reasoning in Intelligent Systems Networks of Plausible Inference*. Morgan Kaufmann.

[34] J. Pearl (1990): *System Z: A Natural Ordering of Defaults with Tractable Applications to Nonmonotonic Reasoning*. In: *TARK'90, Pacific Grove, CA, USA*, pp. 121–135.

[35] D. Porello, O. Kutz, G. Righetti, N. Troquard, P. Galliani & C. Masolo (2019): *A Toothful of Concepts: Towards a Theory of Weighted Concept Combination*. In: *Proceedings of the 32nd International Workshop on Description Logics*, *CEUR Workshop Proceedings* 2373, CEUR-WS.org.

[36] Eleanor Rosch (1973): *Natural Categories*. Cognitive Psychology 4(3), pp. 328–350, doi:*10.1016/0010-0285(73)90017-0*.

[37] M. Wilhelm, A. Thevapalan & G. Kern-Isberner (2023): *Prioritizing Answer Sets Based on Conditional Expert Knowledge*. In: *FLAIRS 2023*, doi:*10.32473/FLAIRS.36.133167*.

Enhancing Compilation-based ASP through Lazy Atom Discovery

Andrea Cuteri Giuseppe Mazzotta Francesco Ricca
University of Calabria
Rende, Italy
andrea.cuteri@unical.it giuseppe.mazzotta@unical.it francesco.ricca@unical.it

Introduction. Answer Set Programming (ASP) [6] is a logic-based formalism based on the stable model semantics [19]. Over the years, several interesting applications for ASP were found ranging from planning [25] to scheduling [11] and many others [14]. Standard ASP systems such as *CLINGO* [17] and *DLV* [2] are based on the ground & solve approach. In the grounding step variables are eliminated from non-ground programs. The result of the grounding process is a propositional-equivalent program which can be evaluated by ASP solvers. Typically, the solving step exploits a CDCL-like algorithm for evaluating variable-free programs. In many cases of practical interest computing variable-free programs might be infeasible [7, 23]. This limitation is known as the grounding bottleneck [18]. Several approaches such as hybrid formalisms [3, 16, 23, 26], compilation-based ASP solving [9, 10, 22, 12], complexity-driven rewriting [4] and lazy grounding [5, 20, 21, 24, 27] were proposed for overcoming the grounding bottleneck. Among these techniques, compilation-based ASP solving revealed to be very promising. Compilation-based solvers transform non-ground (sub)programs into ad-hoc procedures called propagators which are used to mimic inferences coming from the ground ASP program without materializing it. However, compilation-based solvers generate in advance the propositional atoms required for solving the problem (i.e., computing answer sets), and whenever this set of atoms becomes large, these systems exhibit overhead [12]. In this paper, we investigate a novel compilation-based technique that overcomes this limitation by postponing the generation of ground atoms, during the solving phase, only when they need to be included in an answer set. Such *lazy atom discovery* technique allows to reduce the overhead encountered by compilation-based solvers when dealing with a large number of atoms. In what follows, we describe the working principles of lazy atom discovery and report preliminary experimental findings.

Lazy Atom Discovery. Modern ASP solvers, such as CLINGO [16] and DLV2 [1], transform ground rules into clauses by applying the well-known Clark's completion [8]. Such transformation allows to compute answer sets of tight programs (i.e. programs without positive recursion) by leveraging unit propagation of program clauses. However, the Clark's completion requires the introduction of auxiliary atoms which can be detrimental for large programs. For avoiding the materialization of such a translation, it is possible to equip the solver with ad-hoc propagators implementing the Fitting operator [15]. This operator guarantees that rules of a logic program are satisfied and true atoms are also supported w.r.t. a candidate answer set. More precisely, given a program Π and a (partial) interpretation I, the Fitting operator Φ_Π is defined as [15]:

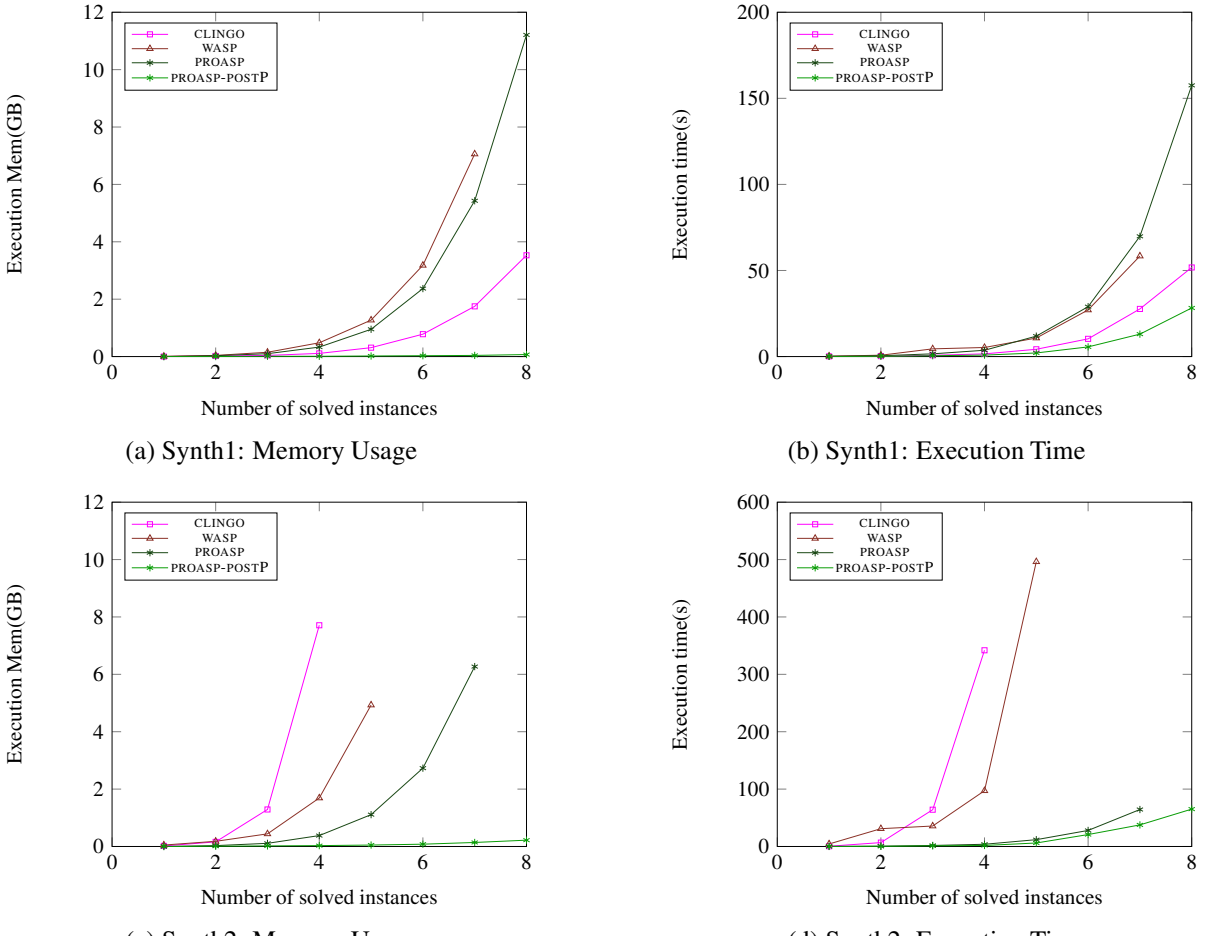
- $\Phi_\Pi(I) = T_\Pi(I) \cup \overline{\gamma_\Pi(I)}$
- $T_\Pi(I) = \{H(r) \mid r \in \mathit{ground}(\Pi) \wedge I \models B(r)\}$
- $\gamma_\Pi(I) = \{a \in B_\Pi \mid \forall r \in \mathit{ground}(\Pi)\ s.t.\ H(r) = a, I \not\models B(r)\}$.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 202–205, doi:10.4204/EPTCS.439.14

© A. Cuteri et al.
This work is licensed under the
Creative Commons Attribution License.

By leveraging the Fitting operator, it is possible to compile a subprogram into ad-hoc propagators that are able to lazily generate atoms derived by the application of the Fitting operator on the compiled subprogram. This strategy is not applicable to arbitrary subprograms, but only to those that we refer to as *lazy program splits*. Intuitively, given a program Π, a stratified subprogram $L \subseteq \Pi$ is a *lazy program split* for Π if it defines predicates that do not appear in the head of any rule of $\Pi \setminus L$. In this way, the solver can search for an answer set of Π without complete knowledge of ground atoms that are defined within a lazy program split. Given a lazy program split L of Π, then it is possible to compile rules in L into two propagators which implement respectively $T_L(I)$ and $\gamma_L(I)$ where I is an arbitrary interpretation. More precisely, the $T_L(I)$ propagator follows a bottom-up evaluation to compute those atoms appearing in the head of some rule in L having true body w.r.t. I. On the other hand, the γ_L propagator follows a top-down evaluation to derive atoms that are no longer supported. It is important to note that the evaluation of lazy program splits cannot be carried out by grounders (i.e., before or after the solving), since predicates appearing in the head of the split L may appear in the body of rules in $\Pi \setminus L$.

Figure 1: Synthetic benchmarks



(a) Synth1: Memory Usage

(b) Synth1: Execution Time

(c) Synth2: Memory Usage

(d) Synth2: Execution Time

Experiments. To assess the impact of such technique we propose a preliminary experiment in which we compare the classical PROASP approach [12, 13] with the one extended with lazy atom discovery. In this comparison we included also state-of-the-art systems CLINGO and WASP. These approaches have been compared on two benchmarks proposed to stress strengths and weaknesses of compilation-based ASP solving [12]. Obtained results are reported in Figure 1. As it can be observed, SYNTH1 highlights the overhead encountered by PROASP when a large number of atoms is required. In this case, Ground & Solve systems perform better than the standard PROASP approach, but the most effective strategy is enabling lazy atom discovery in PROASP. Conversely, SYNTH2 highlights the effectiveness of the standard PROASP approach w.r.t. Ground & Solve approaches by taking a smaller execution time and reducing the overall memory footprint. However, by enabling lazy atom discovery the effectiveness of PROASP is much more remarked. Thus, in conclusion, lazy atom discovery allows to significantly enhance compilation-based ASP solving.

References

[1] Mario Alviano, Francesco Calimeri, Carmine Dodaro, Davide Fuscà, Nicola Leone, Simona Perri, Francesco Ricca, Pierfrancesco Veltri & Jessica Zangari (2017): *The ASP System DLV2*. *LNCS* 10377, Springer, pp. 215–221, doi:10.1007/978-3-319-61660-5_19.

[2] Mario Alviano, Carmine Dodaro, Nicola Leone & Francesco Ricca (2015): *Advances in WASP*. *LNCS* 9345, Springer, pp. 40–54, doi:10.1007/978-3-319-23264-5_5.

[3] Marcello Balduccini & Yuliya Lierler (2017): *Constraint answer set solver EZCSP and why integration schemas matter*. *Theory Pract. Log. Program.* 17(4), pp. 462–515, doi:10.1017/S1471068417000102.

[4] Viktor Besin, Markus Hecher & Stefan Woltran (2022): *Body-Decoupled Grounding via Solving: A Novel Approach on the ASP Bottleneck*. In: *IJCAI*, ijcai.org, pp. 2546–2552, doi:10.24963/IJCAI.2022/353.

[5] Jori Bomanson, Tomi Janhunen & Antonius Weinzierl (2019): *Enhancing Lazy Grounding with Lazy Normalization in Answer-Set Programming*. AAAI Press, pp. 2694–2702, doi:10.1609/AAAI.V33I01.33012694.

[6] Gerhard Brewka, Thomas Eiter & Miroslaw Truszczynski (2011): *Answer set programming at a glance*. *Commun. ACM* 54(12), pp. 92–103, doi:10.1145/2043174.2043195.

[7] Francesco Calimeri, Martin Gebser, Marco Maratea & Francesco Ricca (2016): *Design and results of the Fifth Answer Set Programming Competition*. *Artif. Intell.* 231, pp. 151–181, doi:10.1016/J.ARTINT.2015.09.008.

[8] Keith L. Clark (1977): *Negation as Failure*. In: *Logic and Data Bases*, Advances in Data Base Theory, Plemum Press, New York, pp. 293–322, doi:10.1007/978-1-4684-3384-5_11.

[9] Bernardo Cuteri, Carmine Dodaro, Francesco Ricca & Peter Schüller (2019): *Partial Compilation of ASP Programs*. *Theory Pract. Log. Program.* 19(5-6), pp. 857–873, doi:10.1017/S1471068419000231.

[10] Bernardo Cuteri, Carmine Dodaro, Francesco Ricca & Peter Schüller (2020): *Overcoming the Grounding Bottleneck Due to Constraints in ASP Solving: Constraints Become Propagators*. In: *IJCAI*, ijcai.org, pp. 1688–1694, doi:10.24963/IJCAI.2020/234.

[11] Carmine Dodaro & Marco Maratea (2017): *Nurse Scheduling via Answer Set Programming*. *LNCS* 10377, Springer, pp. 301–307, doi:10.1007/978-3-319-61660-5_27.

[12] Carmine Dodaro, Giuseppe Mazzotta & Francesco Ricca (2023): *Compilation of Tight ASP Programs*. *Frontiers in Artificial Intelligence and Applications* 372, IOS Press, pp. 557–564, doi:10.3233/FAIA230316.

[13] Carmine Dodaro, Giuseppe Mazzotta & Francesco Ricca (2024): *Blending Grounding and Compilation for Efficient ASP Solving*. In: *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR*, doi:10.24963/KR.2024/30.

[14] Esra Erdem, Michael Gelfond & Nicola Leone (2016): *Applications of Answer Set Programming*. *AI Mag.* 37(3), pp. 53–68, doi:10.1609/AIMAG.V37I3.2678.

[15] Melvin Fitting (1985): *A Deterministic Prolog Fixpoint Semantics*. *J. Log. Program.* 2(2), pp. 111–118, doi:10.1016/0743-1066(85)90014-7.

[16] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub & Philipp Wanko (2016): *Theory Solving Made Easy with Clingo 5*. *OASICS* 52, Schloss Dagstuhl, pp. 2:1–2:15, doi:10.4230/OASICS.ICLP.2016.2.

[17] Martin Gebser, Roland Kaminski, Arne König & Torsten Schaub (2011): *Advances in* gringo *Series 3*. In: *LPNMR*, *LNCS* 6645, Springer, pp. 345–351, doi:10.1007/978-3-642-20895-9_39.

[18] Martin Gebser, Nicola Leone, Marco Maratea, Simona Perri, Francesco Ricca & Torsten Schaub (2018): *Evaluation Techniques and Systems for Answer Set Programming: a Survey*. In: *IJCAI*, ijcai.org, pp. 5450–5456, doi:10.24963/IJCAI.2018/769.

[19] Michael Gelfond & Vladimir Lifschitz (1991): *Classical Negation in Logic Programs and Disjunctive Databases*. *New Gener. Comput.* 9(3/4), pp. 365–386, doi:10.1007/BF03037169.

[20] Claire Lefèvre & Pascal Nicolas (2009): *The First Version of a New ASP Solver : ASPeRiX*. *LNCS* 5753, Springer, pp. 522–527, doi:10.1007/978-3-642-04238-6_52.

[21] Yuliya Lierler & Justin Robbins (2021): *DualGrounder: Lazy Instantiation via Clingo Multi-shot Framework*. *LNCS* 12678, Springer, pp. 435–441, doi:10.1007/978-3-030-75775-5_29.

[22] Giuseppe Mazzotta, Francesco Ricca & Carmine Dodaro (2022): *Compilation of Aggregates in ASP Systems*. AAAI Press, pp. 5834–5841, doi:10.1609/AAAI.V36I5.20527.

[23] Max Ostrowski & Torsten Schaub (2012): *ASP modulo CSP: The clingcon system*. *Theory Pract. Log. Program.* 12(4-5), pp. 485–503, doi:10.1017/S1471068412000142.

[24] Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli & Gianfranco Rossi (2009): *GASP: Answer Set Programming with Lazy Grounding*. *Fundam. Informaticae* 96(3), pp. 297–322, doi:10.3233/FI-2009-180.

[25] Tran Cao Son, Enrico Pontelli, Marcello Balduccini & Torsten Schaub (2023): *Answer Set Planning: A Survey*. *Theory Pract. Log. Program.* 23(1), pp. 226–298, doi:10.1017/S1471068422000072.

[26] Benjamin Susman & Yuliya Lierler (2016): *SMT-Based Constraint Answer Set Solver EZSMT (System Description)*. *OASIcs* 52, Schloss Dagstuhl, pp. 1:1–1:15, doi:10.4230/OASICS.ICLP.2016.1.

[27] Antonius Weinzierl (2017): *Blending Lazy-Grounding and CDNL Search for Answer-Set Solving*. *LNCS* 10377, Springer, pp. 191–204, doi:10.1007/978-3-319-61660-5_17.

Defeasible Conditionals using Answer Set Programming

Racquel Dennison
University of Cape Town and CAIR
Cape Town, South Africa
dnnrac003@myuct.ac.za

Jesse Heyninck
Open Universiteit
Heerlen, Limburg, the Netherlands
University of Cape Town and CAIR
Cape Town, South Africa
jesse.heyninck@ou.nl

Thomas Meyer
University of Cape Town and CAIR
Cape Town, South Africa
tmeyer@cair.org.za

Defeasible entailment is concerned with drawing plausible conclusions from incomplete information. A foundational framework for modelling defeasible entailment is the KLM framework. Introduced by Kraus, Lehmann, and Magidor, the KLM framework outlines several key properties for defeasible entailment. One of the most prominent algorithms within this framework is Rational Closure (RC). This paper presents a declarative definition for computing RC using Answer Set Programming (ASP). Our approach enables the automatic construction of the minimal ranked model from a given knowledge base and supports entailment checking for specified queries. We formally prove the correctness of our ASP encoding and conduct empirical evaluations to compare the performance of our implementation with that of existing imperative implementations, specifically the InfOCF solver. The results demonstrate that our ASP-based approach adheres to RC's theoretical foundations and offers improved computational efficiency.

1 Introduction

Formal logic has played a vital role in addressing reasoning challenges in Artificial Intelligence [14]. It enables the precise and structured representation of information, allowing reasoning processes to be modelled algorithmically [25].

Traditionally, reasoning algorithms were monotonic, which meant that adding new information to a domain did not invalidate previously established conclusions [15]. However, monotonicity presented challenges when modelling the characteristics of a domain [22] where incoming information may have contradicted or served as an exception to the existing rules.

Nonmonotonic reasoning (NMR) aims to address these limitations by incorporating mechanisms to handle exceptions [2]. This involves making and revising assumptions previously made when new information is learned. NMR is based on the notion of *typicality*, which means that not all properties of a given object necessarily hold [7]. For example, instead of asserting that "all birds fly", we say that "birds *typically* fly" as there are exceptional cases such as ostriches and penguins. By defining the characteristics of an object to "*typically*" hold, we make provisions for exceptions that could arise.

Defeasible reasoning offers a framework for modelling this type of reasoning. Among the various approaches to defeasible reasoning, the KLM framework developed by Kraus, Lehmann, and Magidor [16] is well-known for defining the fundamental properties any defeasible reasoning framework should follow when dealing with incomplete information [18].

A notable reasoning pattern that satisfies the KLM properties is Rational Closure (RC). RC is a conservative form of defeasible reasoning, meaning that it infers minimal assumptions from the knowledge base [4]. RC is defined both semantically and algorithmically.

While both KLM and ASP are used to model nonmonotonic reasoning, they make use of different paradigms when it comes to modelling nonmontonic reasoning. The KLM framework adopts a prefer-

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 206–223, doi:10.4204/EPTCS.439.15

© Racquel Dennison, Jesse Heyninck & Thomas Meyer
This work is licensed under the
Creative Commons Attribution License.

ential semantics approach, ranking all possible worlds according to a preference relation. A formula α is considered satisfiable if it holds in all of the most preferred models [27]. ASP employs negation as failure, a form of default reasoning in which a formula α is assumed to be false if it cannot be proven to be true by the given rules within the program. This distinction highlights a key divergence in how each system handles uncertainty and incomplete information.

Most of the existing approaches for computing entailment relations that satisfy the KLM properties have been imperative [4], with some work exploring declarative programming techniques [12].

In this paper, we present an alternative declarative definition of RC using ASP. We provide the encoding of RC and prove the correctness of our definitions. Furthermore, we present our findings from testing the computational runtime of our RC definition. Through extensive experiments, we show that our ASP-based implementation outperforms the InfOCF solver [1].

2 Background

In this section, we provide the relevant background material. Propositional logic is presented in Section 2.1. Defeasible reasoning and the KLM approach are presented in Section 2.2. Rational Closure and Answer Set Programming are presented in Section 2.3 and 2.4.

2.1 Propositional Logic

The finite set of all propositional atoms is denoted by $\mathcal{P}$. Propositional atoms are assigned a *truth value*, either true or false [23], and are combined with boolean operators to create propositional statements, otherwise known as formulas. Formulas are denoted by Greek letters such as α, β, and γ. The language, denoted as $\mathcal{L}$, is a set of formulas which are represented as, for any given $p \in \mathcal{P}$ and $\tau, \alpha \in \mathcal{L}$, τ can take on the following form: $p, \neg\tau, \tau \wedge \alpha, \tau \rightarrow \alpha, \tau \leftrightarrow \alpha, \tau \vee \alpha$ [23]. A knowledge base, $\mathcal{K} \subseteq \mathcal{L}$, is a set of propositional formulas which can be finite or infinite. In this paper, we only consider finite knowledge bases. We define an interpretation ω, as a mapping from propositional atoms to truth values. An interpretation, ω is defined as $\omega : \mathcal{P} \rightarrow \{T, F\}$ [23]. We denote the value of a formula under an interpretation as $\mathcal{I}(\alpha)$. If $\mathcal{I}(\alpha)$ is true for some formula α, we say that $\mathcal{I}$ satisfies α, denoted as $\mathcal{I} \Vdash \alpha$. If at least one interpretation, $\mathcal{I}$, satisfies $\mathcal{K}$ then $\mathcal{I}$ is a *model* of $\mathcal{K}$. The set of all models of $\mathcal{K}$ is denoted as $\text{mod}(\mathcal{K})$. If there exists no $\mathcal{I} \in \mathcal{U}$ such that $\mathcal{I} \Vdash \mathcal{K}$, then $\mathcal{K}$ contains no interpretation satisfying all $\alpha \in \mathcal{K}$. If there is no $\mathcal{I}$ such that $\mathcal{I} \Vdash \mathcal{K}$, $\mathcal{K}$ is said to be unsatisfiable. Drawing inferences from a knowledge base is defined as entailment, denoted as $\models$.

2.2 Defeasible Reasoning

Defeasible reasoning is a nonmonotonic framework that allows the retraction of previously established rules when new, contradictory information is introduced to a knowledge base [1]. Various formulations have been developed to model defeasible reasoning, including belief revision [24], default logic [11], and the preferential approach [26]. The preferential approach has been extended into what is known as the KLM framework [16]. This framework is the paper's primary focus due to its well-defined properties and computationally efficient reasoning algorithms, one of which is rational closure (in short, RC) [4].

Our work is situated in the *KLM framework*, which extends the propositional language by introducing defeasible implications [18]. A defeasible implication is denoted by $\mid\sim$, expressed as $\alpha \mid\sim \beta$ where $\alpha, \beta \in$

[1] https://github.com/jonasphilipp/InfOCF

$\mathcal{L}$. We thus define the language $\mathcal{L}_p = \mathcal{L} \cup \{\alpha \mid\sim \beta \mid \alpha, \beta \in \mathcal{L}\}$ to be an extension of the propositional language, $\mathcal{L}$, with the added defeasible connective. This allows us to form formulas using $\mid\sim$. Note that we do not allow nesting of $\mid\sim$.

Drawing inferences from a set of defeasible implications is called defeasible entailment. The notion of defeasible entailment ($\mid\approx$) is not unique; many different ways of defining defeasible entailment have been outlined. The KLM definition of defeasible entailment is of interest in this paper due to its properties, the KLM properties [16], and well-defined algorithms associated with it.

2.3 Rational Closure

RC is a conservative form of defeasible reasoning, which means that the reasoning algorithm infers as minimally as possible from a knowledge base [15] but allows for handling exceptional information. The framework is defined semantically as well as algorithmically [4]. This paper will focus on the algorithmic definition of RC as outlined by Casini et al. [4]. Computing defeasible entailment with RC requires two algorithms, BaseRank and RC.

Algorithm 1 BaseRank

Input: A knowledge base $\mathcal{K}$
Output: An ordered tuple (R_0,..., R_{n-1},R_∞,n). Each R_i, where $i \in [0, n]$ indicates the rank each statement is placed on.
$i = 0$;
$E_0 = \overrightarrow{\mathcal{K}}$;
while $E_{i-1} \neq E_i$ **do**
    $E_{i+1} = \{ \alpha \rightarrow \beta \in E_i | E_i \models \neg\alpha \}$;
    $R_i = E_i \setminus E_{i+1}$;
    $i = i + 1$;
end while
$R_\infty = E_{i-1}$;
if $E_{i-1} = \varnothing$ **then**
    $n = i - 1$;
else
    $n = i$;
end if
return (R_0,..., R_{n-1},R_∞,n);

BaseRank is a procedure which assigns a numerical rank to statements in a knowledge base based on their exceptionality. General statements, which apply broadly to a particular topic, are given lower ranks, while more exceptional statements which violate established rules within a knowledge base are assigned higher ranks. For example, given the knowledge base $\mathcal{K} = \{\text{man} \mid\sim \text{mortal}, \text{Socrates} \rightarrow \text{man}, \text{Socrates} \rightarrow \neg\text{mortal}\}$, the statement Socrates $\rightarrow \neg$mortal is exceptional while man $\mid\sim$ mortal is a general statement applicable to the class *man*.

The BaseRank procedure performs ranking on a knowledge base which has been transformed to only classical material implications. We refer to this as materialisation, formalised in Definition 1. Algorithm 1 outlines the BaseRank procedure.

Definition 1. The materialisation of a knowledge base $\mathcal{K}$ is defined as $\overrightarrow{\mathcal{K}} := \{\alpha \rightarrow \beta \mid \alpha \mid\sim \beta \in \mathcal{K}\}$ [15].

Definition 2. A formula α is exceptional in the set $\mathcal{K}$ if $\mathcal{K} \models \neg\alpha$ [15].

RC determines whether a knowledge base entails an implication. For a given query $\alpha \mid\sim \beta$, α is the antecedent and β is the consequence. To compute defeasible entailment for a given query, $\alpha \mid\sim \beta$, RC takes a defeasible knowledge base, computes BaseRank, then repeatedly eliminates ranks assigned to statements within the knowledge base which make the antecedent exceptional, eventually settling on the maximal subset $\overrightarrow{\mathcal{K}'}$ of $\overrightarrow{\mathcal{K}}$ which does not make α exceptional. RC returns true to a query if $\alpha \rightarrow \beta$ is classically entailed by $\overrightarrow{\mathcal{K}'}$. Entailment under RC is denoted as $\mathcal{K} \mid\approx_{RC}$. Algorithm 2 outlines the RC procedure.

Algorithm 2 Rational Closure

Input: A knowledge base $\mathcal{K}$ and a defeasible implication $\alpha \mid\sim \beta$.
Output: true, if the query is entailed by the knowledge base, else **false**.
$(R_0,..., R_{n-1},R_\infty,n) = BaseRank(\mathcal{K})$;
$i = 0$;
$\mathcal{R} = \bigcup_{j=0}^{j<n-1} \mathcal{R}_j$;
while $\mathcal{R}_\infty \cup \mathcal{R} \models \neg\alpha$ *and* $\mathcal{R} \neq \varnothing$ **do**
    $\mathcal{R} = \mathcal{R} \setminus \mathcal{R}_i$;
    $i = i + 1$;
end while
return $\mathcal{R}_\infty \bigcup \mathcal{R} \models \alpha \rightarrow \beta$;

Example 1 Given the knowledge base $\mathcal{K}_1 = \{\text{man} \mid\sim \text{mortal}, \text{Socrates} \rightarrow \text{man}, \text{Socrates} \rightarrow \neg\text{mortal}\}$ and the query, Socrates $\rightarrow$ man. The antecedent of the query is Socrates. The materialisation of $\mathcal{K}_1$ is $\{\text{man} \rightarrow \text{mortal}, \text{Socrates} \rightarrow \text{man}, \text{Socrates} \rightarrow \neg\text{mortal}\}$ and the base rank for $\mathcal{K}_1$ is outlined in Table 1.

TABLE 1: BaseRank of knowledge base $\mathcal{K}_1$

$\mathcal{R}_\infty$	Socrates $\rightarrow$ man, Socrates $\rightarrow \neg$mortal
$\mathcal{R}_0$	man $\rightarrow$ mortal

TABLE 2: BaseRank of knowledge base $\mathcal{K}_1$ with the lowest rank eliminated.

$\mathcal{R}_\infty$	Socrates $\rightarrow$ man, Socrates $\rightarrow \neg$mortal
$\mathcal{R}_0$	~~man $\rightarrow$ mortal~~

To determine if $\mathcal{R} \cup \mathcal{R}_\infty \models \neg$Socrates; models of $\mathcal{K}_1$ are computed. We denote Socrates as s, man as m, and mortal as w. The models of $\mathcal{K}_1$ are $mod(\mathcal{K}_1) = \{\overline{s}mw, \overline{sm}w, \overline{smw}\}$. We note that for all interpretations of $\mathcal{K}_1$ $\neg$Socrates holds, $\mathcal{R} \cup \mathcal{R}_\infty \models \neg$Socrates. RC will thus eliminate rank 0. The revised models of $\mathcal{K}_1$ are $\{\overline{s}mw, \overline{sm}w, \overline{smw}, sm\overline{w}\}$. We see that $\mathcal{R} \cup \mathcal{R}_\infty \not\models \neg$Socrates as there exists an interpretation which makes Socrates true. Computing classical entailment on $\mathcal{R} \cup \mathcal{R}_\infty$, we see that at least one model within the remaining ranks makes the statement *Socrates* $\rightarrow$ *man* true, thus under RC, $\mathcal{K} \mid\approx_{RC}$ Socrates $\rightarrow$ *man*. This example highlights the use of RC. Under classical logic, it would have been concluded that Socrates does not exist. RC allowed us to capture the exceptional case of Socrates and represent this within a knowledge base.

2.4 Answer Set Programming

ASP is a modern approach to declarative programming designed to solve complex problems [21]. It is recognised for its ease of use, expressive constructs, and computational efficiency [8]. ASP consists of rules similar to those in Prolog [5] and has computational methods that leverage the concepts behind efficient satisfiability solvers [8]. Solving ASP programs involves determining stable models [20]. A stable model is a set of atoms which satisfy the rules specified in an ASP program. Problems are encoded into programs that can be processed by ASP tools like *gringo* [10] and *clasp* [9]. The solver used in this paper is *clasp* [9]. *Clingo* [8] combines both *gringo* and *clasp* into a monolithic system and is a programming tool based on the ASP programming paradigm. Rules in ASP take the form: $a_0 \leftarrow a_1, a_2, ..., a_m, \sim a_{m+1}, .., \sim a_n$. We denote its head as $\text{head}(r) = \{a_0\}$. The body is defined as $\text{body}(r) = \{a_1, a_2, \ldots, a_m, \sim a_{m+1}, \ldots, \sim a_n\}$. Both the head and the body of the rule consist of literals. Literals are atoms that are either positive or negative [8]. The set of positive literals in the body is denoted by $body^+(r) = \{a_1, a_2, ..., a_m\}$ while the set of negative literals is $body^-(r) = \{a_{m+1}, ..., a_n\}$. Intuitively, if we have a set $\mathcal{X}$ which consists of literals within the program then if $body^+(r) \subseteq \mathcal{X}$ and $body^-(r) \cap \mathcal{X} = \varnothing$ then $head(r) \in \mathcal{X}$ [8]. Answer sets are minimal supported models of an ASP program. The full definition can be found in [8].

Choice rules describe several alternative ways to form a stable model, with the head of the rule consisting of an expression in braces [8]. For example: $\{a; b\}$

states all possible ways the two literals a and b can be included in the stable model. There are four possible choices, which are the power set of $\{a, b\}$, namely, $\{\{\varnothing\}, \{a\}, \{b\}, \{a, b\}\}$.

ASP allows for expressing cost functions subject to minimisation and/or maximisation. This is especially useful when solving optimisation problems. The function we are interested in is the *minimize statement*, expressed as $minimize\{l_1 = w_1@p_1, \ldots, l_n = w_n@p_n\}$. The minimize directive instructs the solver to compute the optimal answer set that minimises the elements' weighted sum, $l_1 = w_1@p_1, \ldots, l_n = w_n@p_n$.

3 Declarative approach to BaseRank and Rational Closure

In this section, we present our definition of BaseRank (Section 3.1) and RC (Section 3.2). In each, we describe the structure of the problem instance and its corresponding problem encoding in ASP.

3.1 BaseRank

We describe classical implications ($\alpha \rightarrow \beta$) and defeasible implications ($\alpha \mid\sim \beta$) by the following predicates, `classical/2` and `defeasible/2`. These predicates serve as the problem instances for BaseRank. Listing 1 shows how we would view the knowledge base $\mathcal{K}_1$ as presented in Example 1.

```
1 defeasible(man,mortal).
2 classical(socrates,man).
3 classical(socrates,-mortal).
```

LISTING 1: Encoding of knowledge base $\mathcal{K}_1$

The problem encodings for BaseRank are solved via schematic rules. Schematic rules contain variables and allow for the generalisation of logic rules [8]. Our definition of BaseRank in Listing 2 follows the *generate, define and test* methodology [19].

The preliminary rules on lines 2 and 3 convert the knowledge base encodings to material implications, defined by the predicate `m_implication/2`. All classical rules are then ranked on the infinite rank, as described by Line 4.

To ensure the knowledge base itself is not inconsistent, we define the predicate `infer/2` on lines 5,6 and 7. Any knowledge base which derives $\top \rightarrow \alpha$ and $\neg\alpha \rightarrow \alpha$ would be unsatisfiable; thus, the `infer/2` predicate allows us to define constraints to catch these cases and define a global state of the knowledge base.

The *generate* section on line 8 arbitrarily assigns a numerical rank to every material implication using a choice rule. The predicate `rank(m_implication(X,Y),N)` expresses that the material implication, $X \rightarrow Y$ encoded as `m_implication(X,Y)` is assigned a numerical rank `N`. The choice rule has a cardinality constraint of one. This ensures that for every solution, a material implication is placed on one rank.

In the worst case, each statement will be placed on a unique rank. Thus, the choice rule on line 8 assigns each material implication to an arbitrary rank, where the maximum rank a statement can be placed on is based on the number of statements within a knowledge base. For instance, if we consider $\mathcal{K}_1$, in the worst case, each statement would be assigned a unique rank, meaning there are 3 ranks for the ranked model.

The *define* section of the encoding is from lines 9 to 12. This section derives all the possible consequences within the knowledge base. The `derive/3` predicate encodes the natural deductive rules for transitivity, identity and disjunctive elimination. The predicate `derive(X,Y,N)` is defined as $X \rightarrow Y$ holds on rank `N`.

The *test* section of the encoding is from lines 14 to 19. Line 14 ensures that if an answer set contains the predicate `derive(X,Y,N)` and the `derive(X,-Y,N)`, the answer set is rejected. Line 15 ensures that any antecedent which is a tautology should not have the negation of itself present on that rank. Lines 18 and 19 ensure that inconsistent knowledge bases are rejected.

Lines 21 to 23 make use of the `#minimize` optimisation directive to ensure all implications are assigned the lowest ranks possible while preserving the distribution structure of implications; furthermore, all antecedents which are tautologies are assigned the lowest ranks possible.

```
1  m_implication(X,Y) :- defeasible(X,Y).
2  m_implication(X,Y) :- classical(X,Y).
3  statement_count(C):- C = #count{X,Y : m_implication(X,Y)}.
4  rank(m_implication(X,Y),inf) :- classical(X,Y).
5  infer(X,Y) :- rank(m_implication(X,Y),N).
6  infer(X,P) :- rank(m_implication(Y,P),N1), infer(X,Y).
7  infer(top,Y) :- infer(X,Y), infer(-X,Y).
8  {rank(m_implication(X,Y),0..C-1)} =1 :- statement_count(C), m_implication(X,Y),
       not classical(X,Y).
9  derive(X,X,N) :- rank(m_implication(X,Y),N).
10 derive(X,Y,N) :- rank(m_implication(X,Y),N), derive(X,X,N).
11 derive(X,P,N) :- rank(m_implication(Y,P),N1), derive(X,Y,N), N1 >= N.
12 derive(top,Y,N) :- derive(X,Y,N1), derive(-X,Y,N) , N1 >= N.
13
14 :- derive(X,Y,N), derive(X,-Y,N).
15 :- derive(-X,_,N), derive(top,X,N).
16 :- derive(_,-X,N), derive(top,X,N).
17 :- rank(m_implication(X,_),N1), rank(m_implication(X,_),N2), N1 != N2, not
       classical(X,_), not coded_classical(X,_).
18 :- infer(top,X), infer(-X,X).
19 :- infer(top,Z), infer(top,X), infer(Z,-X).
```

```
20
21 #minimize{N,X,Y : rank(m_implication(X,Y),N)}.
22 #minimize{N,Y : derive(top,Y,N)}.
23 #minimize{N,Y,X : derive(top,Y,N),rank(m_implication(-X,Y),N) , rank(
       m_implication(X,Y),N)}.
24 #show rank/2.
```

LISTING 2: Problem encoding of BaseRank

The computed ranked model for $\mathcal{K}_1$ using the BaseRank definition as defined in Listing 2 is shown in Listing 3.

3.2 Rational Closure

A query is defined via the predicate `query(X,Y)`, while the ranked levels of the statements obtained from computing BaseRank are described via the predicate `rank(m_implication(X,Y),N)`. Our definition of RC can handle queries of the form $p \mid\sim q$ and p where p, q are literals and contain no boolean connectives.

```
1 rank(m_implication(socrates,man),inf).
2 rank(m_implication(socrates,-mortal),inf).
3 rank(m_implication(man,mortal),0).
```

LISTING 3: BaseRank model of $\mathcal{K}_1$

Similar to BaseRank, the definition of RC follows the *generate, define and test* approach.

In Line 1 of Listing 4, we generate the predicate `guess/1` by means of a choice rule. This rule states that for all the rank levels `N` that exist in the `rank(_,N)` facts, choose exactly one guess(N). The `guess/1` predicate acts as an arbitrary rank chosen from the ranked model.

The predicate `inference/2` encodes material implications that can be inferred by the knowledge base. These are outlined by the rules in Lines 4-8. Line 4 assumes some interpretation exists which makes the antecedent `X` true; this follows to be the identity rule of natural deduction; thus, the fact `inference(X,X)` is added to the answer set. Line 6 makes use of the transitive rule for material implications, and Lines 8 and 9 capture all logical inferences from the material implications by making use of contrapositive rules.

```
1 {guess(N):rank(_,N)}=1.
2 inference(X,X) :- rank(m_implication(X,Y),N1), guess(N), N<=N1.
3 inference(X,Y) :- rank(m_implication(X,Y),N1), guess(N), N<=N1.
4 inference(X,Z) :- inference(X,Y), rank(m_implication(Y,Z),N1), guess(N), N<=N1.
5 inference(-Y,-X) :- rank(m_implication(X,Y),N1), guess(N), N<=N1.
6 inference(-Z,-X) :- inference(X,Y), rank(m_implication(Y,Z),N1), guess(N), N<=N1
      .
7 inference(top,Y) :-inference(X,Y), inference(-X,Y).
8 inference(top,-X) :-inference(X,Y), inference(X,-Y).
9 :- query(X,Y), inference(top, -X).
10 entailed(true):- inference(X,Y), query(X,Y).
11 entailed(false) :- not entailed(true).
12 #minimize{N:guess(N)}.
13 #show entailed/1.
```

LISTING 4: Problem encoding of Rational Closure

Lines 7 and 8 play vital roles in determining which ranks satisfy the query antecedent. Both lines encode disjunctive elimination. For line 7, if we have some inference, for example, `inference(X,Y)` and `inference(-X,Y)` then it is always true that the consequence, `Y`, holds. Line 8 states that if we can derive `inference(X,Y)` and `inference(X,-Y)` then $-X$ is true, making X false.

The inference rules correspond to standard natural deduction rules, namely, identity, modus ponens, contrapositive, and disjunctive elimination.

Line 9 rejects any answer set where the antecedent of a query is exceptional in a guessed rank. Entailment is defined on line 10. This definition states that if `inference(X,Y)` is defined in a given answer set and `query(X,Y)` also holds, then the query is entailed by the knowledge base. A statement is not entailed by a knowledge base when there is no evidence in the answer set to suggest that `entailed(true)` holds.

The `#minimize` directive ensures the smallest rank, which does not result in an exceptional antecedent, is chosen.

Example 2 We begin by defining a simple knowledge base, $\mathcal{K}_1$, shown in Listing 5. This includes defeasible and classical conditionals as well as a query. The knowledge base is encoded in a file called `knowledge_base.lp`.

```
1 defeasible(man,mortal).
2 classical(socrates,man). classical(socrates,-mortal).
3 query(socrates,man).
```

LISTING 5: Encoding of knowledge base $\mathcal{K}_1$

To evaluate the query using the RC procedure, we run the following command in our terminal `clingo knowledge_base.lp base_rank.lp rational_closure.lp`. The resulting optimal answer set is shown in Listing 6. The output given shows that only one answer set was computed by the solver. We observe that the rank guess was the infinite rank, which led to the inferences shown in the answer sets. Given that the atom `inference(socrates,man)` is included in the answer set using the inference rules, the query `query(socrates,man)` is entailed by the knowledge base.

```
1 query(socrates,man) guess(inf)
2 inference(socrates,socrates) inference(-man,-socrates)
3 inference(mortal,-socrates) inference(socrates,man)
4 inference(socrates,-mortal) entailed(true)
```

LISTING 6: RC entailment results using Clingo for $\mathcal{K}_1$

3.3 Soundness and Correctness

To prove the correctness of our ASP definitions of BaseRank and RC, we prove the soundness and completeness of each.

3.3.1 BaseRank

First, some notational preliminaries. We denote, given a knowledge base $\mathcal{K}$, `encoding`($\mathcal{K}$) be its encoding (see Section 1) and `baseRank` the program in Listing 2 (excluding the minimize-directive on lines

22-24). We denote the minimize-directive on lines 22-24 of Listing 2 by `baseRankMin`. Furthermore, for a given answer set $\mathcal{S}$ of $\texttt{encoding}(\mathcal{K}) \cup \texttt{baseRank}$, we define $\mathcal{K}_i^{\mathcal{S}}$ and $\overrightarrow{\mathcal{K}}_i^{\mathcal{S}}$ as follows:

$$\mathcal{K}_i^{\mathcal{S}} = \{A \mid\sim B \in \mathcal{K} \mid \texttt{rank(m_implication(A,B),i)} \in \mathcal{S}\}$$

$$\overrightarrow{\mathcal{K}}_i^{\mathcal{S}} = \{A \rightarrow B \mid A \mid\sim B \in \mathcal{K}_i^{\mathcal{S}}\}$$

The following shows that the derived rules encoded in Base rank are exactly those rules used in natural deduction [6].

Proposition 1. Let i be a fixed nonnegative integer and $\mathcal{S}$ be some answer set satisfying the rules outlined in `BaseRank` without the minimize directive. Further define

$$\text{ND}_N = \left\{ X \rightarrow Y \;\middle|\; \overrightarrow{\mathcal{K}}_i^{\mathcal{S}} \models X \rightarrow Y \right\}$$

and

$$\text{ASP}_i = \left\{ X \rightarrow Y \;\middle|\; \texttt{derive(X,Y,i)} \in \mathcal{S} \right\}.$$

Then

$$\text{ND}_i = \text{ASP}_i.$$

The following Proposition shows that the rankings assigned by `baseRank` respect the notion of tolerance or exceptionality used in the BaseRank-algorithm:

Proposition 2. Let a knowledge base $\mathcal{K}$ be given. For a given answer set $\mathcal{S}$ of $\texttt{encoding}(\mathcal{K}) \cup \texttt{baseRank}$, for any $i \geq 0$, and for any $A \mid\sim B \in \mathcal{K}_i^{\mathcal{S}}$, $\bigcup_{j=i}^{\infty} \overrightarrow{\mathcal{K}_j^{\mathcal{S}}} \not\models \neg A$.

Furthermore, `baseRank` is sound in the sense that every "refinement" of the base ranks of $\mathcal{K}$ will occur as an answer set of $\texttt{encoding}(\mathcal{K}) \cup \texttt{baseRank}$:

Proposition 3. Let a knowledge base $\mathcal{K}$ with $\texttt{BaseRank}(\mathcal{K}) = (\mathcal{R}_0, \ldots, \mathcal{R}_n, \mathcal{R}_\infty)$ be given. For any partition $(\mathcal{R}'_0, \ldots, \mathcal{R}'_m, \mathcal{R}_\infty)$ of $\mathcal{K}$ s.t. for every $\mathcal{R}'_i$ there is a j s.t. $\mathcal{R}'_i \subseteq \mathcal{R}_j$, there is an answer set $\mathcal{S}$ of $\texttt{encoding}(\mathcal{K}) \cup \texttt{baseRank}$ s.t. $\mathcal{K}_i^{\mathcal{S}} = \mathcal{R}'_i$ for every $i \geq 0$.

Finally, we show that the minimal answer set of the `baseRank`-encoding coincides with the partition generated by the BaseRank algorithm:

Proposition 4. Let a knowledge base $\mathcal{K}$ with $\textit{BaseRank}(\mathcal{K}) = (\mathcal{R}_0, \ldots, \mathcal{R}_n, \mathcal{R}_\infty)$ be given. For the answer set $\mathcal{S}$ of $\texttt{encoding}(\mathcal{K}) \cup \texttt{baseRank} \cup \texttt{baseRankMin}$, $\mathcal{K}_i^{\mathcal{S}} = \mathcal{R}_i$ for $i \geq 0$.

3.3.2 Rational Closure

We denote the Rational Closure ASP algorithm as `RC` and Π as $\texttt{encoding(K)} \cup \texttt{baseRank} \cup \texttt{RC}$. Furthermore, for any given answer set $\mathcal{S}$ we denote $\bigcup_{j=n}^{\infty} \overrightarrow{\mathcal{K}_j^{\mathcal{S}}} = \{$ A $\mid\sim$ B $\mid \texttt{rank(m_implication(A,B),i)} \in \mathcal{S}\}$

We first show that our encoding is sound:

Proposition 5. Given some knowledge base $\mathcal{K}$ and query A$\mid\sim$ B, if $\texttt{entailed(true)} \in \mathcal{S}$, where $\mathcal{S}$ is the minimal answer set, then $\mathcal{K} \mid\approx_{RC}$ A $\mid\sim$ B.

Likewise, the encoding is complete:

Proposition 6. Given some knowledge base $\mathcal{K}$ and query A$\mid\sim$ B, if $\mathcal{K} \mid\approx_{RC}$ A $\mid\sim$ B then there exists a minimal answer set $\mathcal{S}$ of Π where $\texttt{entailed(true)} \in \mathcal{S}$.

4 Experiments and Results

To assess the computational efficiency of our RC definition, we conducted various experiments testing the runtime of answering queries from a knowledge base. The knowledge bases used consisted of formulas of the form $\alpha \mid\sim \beta$. The queries were statements explicit in the knowledge base. The following section outlines the experimental process and the tests run. We tested our implementation against the InfOCF solver's implementation of System Z.

Experiments were conducted to determine the runtime required to compute answer sets when using our ASP RC definition for entailment. Tests were conducted on a MacBook Air with an M1 processor, featuring an eight-core CPU and a seven-core GPU. The timing for computing entailment using our definition of RC was captured from the statistics captured by the Clasp solver once the process was complete. To compute the timing of RC within the InfOCF solver, we tested the time taken to compute inference using the time function from the Python library. It is important to note that the version of the InfOCF solver tested against is written using Python 3.

Our experiments consist of two variables, the size of a knowledge base and the time taken to compute inference on the knowledge base. We tested knowledge bases starting from 25 statements to 325 statements in increments of 25. For each knowledge base, we tested the time taken to compute the inference of a specified query. Furthermore, we ran 2000 independent trials to represent the time taken accurately. Testing was halted beyond 325 statements due to a recursion error in the InfOCF solver.

Figure 1 presents the average runtime (±95% CI) comparing our ASP-based Rational Closure solver with the InfOCF solver implementation, across knowledge base sizes ranging from 25 to 325 statements. The ASP RC implementation consistently demonstrates significantly lower runtime across all KB sizes, clearly highlighting its computational efficiency. Notably, the runtime difference between the two solvers widens as the KB size increases.

The InfOCF solver shows a linear increase in runtime proportional to KB size, whereas our method's runtime scales significantly better as the size of the knowledge base increases. The least square regression of the ASP solver is 0.00003x + 0.0013, while the InfOCF solver is 0.00035x + 0.0050. Comparing the gradients of each, we notice that our method runs 12 times faster.

The confidence intervals (CIs) further illustrate the efficiency difference: ASP's CI remains consistent at ±1 ms even at the largest tested KB size (325), indicating a stable and predictable performance. In contrast, the InfOCF solver's CI expands to approximately ±3 ms around 275 statements, reflecting greater sensitivity in runtime.

Quantitatively, at the largest KB size, our method achieves a runtime of approximately 0.011 seconds compared to the InfOCF solver's 0.12 seconds, representing roughly a 91% reduction in computational time. These findings indicate that the ASP-based RC solver offers both greater efficiency and improved scalability for entailment reasoning tasks.

Beyond computational performance, the declarative nature of ASP itself presents substantial practical advantages. ASP's clear syntax, consisting of logical implications, makes programs highly readable once the basic logical operations are understood.

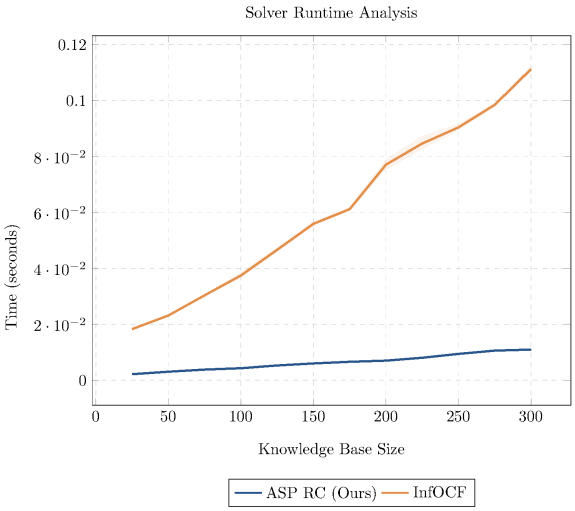


FIGURE 1: Comparison of our ASP implementation of RC and the InfOCF solver

Furthermore, ASP encodings of RC allow ease of modification and extension. For instance, extending our RC solver to support Lexicographic Closure requires only adjustments within the ranking module without requiring a complete redesign of the rules of the program. Additionally, the transparency of ASP allows for straightforward insight into the reasoning process. Since ASP rules directly represent logical specifications, it is immediately clear how answer sets are derived from given premises. This facilitates an easier debugging process and improves long-term maintainability.

5 Conclusion and Future Work

This paper proposed a novel declarative definition for computing entailment under RC framework using ASP. We outlined the underlying intuition for our approach and provided a formal proof of the correctness of both BaseRank and RC, defined using ASP. Additionally, we conducted experimental evaluations comparing our implementation with the InfOCF solver, an imperative alternative for computing RC. The results demonstrated that our ASP-based definition significantly outperformed the InfOCF solver in terms of computational runtime.

Beyond a decrease in computational runtime, we found that ASP has several other advantages for its use within the KLM framework. One lies in its declarative nature, which emphasises the what of a problem rather than the how. We found that this abstraction encouraged a deeper conceptual understanding of RC. In modelling RC, the declarative paradigm not only led to a clearer specification of the problem but also offered a fresh perspective on the underlying logic of KLM entailment relations. Another benefit is the compact and readable nature of the encoding.

The definition of RC presented in this paper considers works with knowledge bases consisting of statements of the form $\alpha \mid\sim \beta$ where α and β are literals. Future work could extend this by exploring more complex logical statements involving conjunctions and disjunctions. Furthermore, while RC is one of the more well-known defeasible forms of reasoning, there are many other forms of defeasible entailment outlined by the KLM framework that can be investigated using ASP in future work. These include Lexicographic Closure [17] and Relevant Closure [3].

Acknowledgements

The authors wish to thank the anonymous reviewers for their helpful comments. This work is based on the research supported in part by the National Research Foundation of South Africa (REFERENCE NO: SAI240823262612). Jesse Heyninck was partially supported by the project LogicLM: Combining Logic Programs with Language Model with the number NGF.1609.241.010 of the research programme NGF AiNed XS Europa 2024-1, which is (partly) financed by the Dutch Research Council (NWO).

References

[1] G. Antoniou, D. Billington, G. Governatori, M. J. Maher & A. Rock (2000): *A family of defeasible reasoning logics and its implementation*. In: *Proceedings of the 14th European Conference on Artificial Intelligence*, ECAI'00, IOS Press, NLD, p. 459–463.

[2] Gerhard Brewka, Ilkka Niemelä & Mirosław Truszczyński (2008): *Chapter 6 Nonmonotonic Reasoning*. In Frank van Harmelen, Vladimir Lifschitz & Bruce Porter, editors: *Handbook of Knowledge Representation*, *Foundations of Artificial Intelligence* 3, Elsevier, pp. 239–284, doi:https://doi.org/10.1016/S1574-6526(07)03006-4.

[3] Giovanni Casini, Thomas Meyer, Kodylan Moodley & Riku Nortjé (2014): *Relevant Closure: A New Form of Defeasible Reasoning for Description Logics*. In Eduardo Fermé & João Leite, editors: *Logics in Artificial Intelligence*, Springer International Publishing, Cham, pp. 92–106, doi:https://doi.org/10.1007/978-3-319-11558-0_7.

[4] Giovanni Casini, Thomas Meyer & Ivan Varzinczak (27-29 October 2018): *Defeasible Entailment: from Rational Closure to Lexicographic Closure and Beyond*. In: *17th International Workshop on Non-Monotonic Reasoning (NMR) 2018*, Arizona, USA, pp. pp. 109–118. Available at https://researchspace.csir.co.za/items/438002eb-e5fe-49a7-b46c-183e19697404.

[5] Alain Colmerauer (1990): *An introduction to Prolog III*. *Commun. ACM* 33(7), p. 69–90, doi:10.1145/79204.79210.

[6] Amr Elnashar & Wafik Boulos Lotfallah (2018): *Minimal Complete Propositional Natural Deduction Systems*. *Journal of Philosophical Logic 47* 47(4), p. 803–815, doi:https://doi.org/10.1007/s10992-017-9450-1.

[7] Dov M. Gabbay & John Woods (January 2, 2014): *Handbook of the History of Logic*. North Holland.

[8] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2012): *Answer Set Solving in Practice*. Morgan and Claypool Publishers.

[9] Martin Gebser, Benjamin Kaufmann, André Neumann & Torsten Schaub (2007): *Clasp: a conflict-driven answer set solver*. In: *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, LPNMR'07, Springer-Verlag, Berlin, Heidelberg, p. 260–265, doi:https://doi.org/10.1007/978-3-540-72200-7_23.

[10] Martin Gebser, Torsten Schaub & Sven Thiele (2007): *GrinGo: a new grounder for answer set programming*. In: *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning*, LPNMR'07, Springer-Verlag, Berlin, Heidelberg, p. 266–271, doi:https://doi.org/10.1007/978-3-540-72200-7_24.

[11] Hector Geffner & Judea Pearl (1992): *Conditional entailment: bridging two approaches to default reasoning*. *Artif. Intell.* 53(2–3), p. 209–244, doi:https://doi.org/10.1016/0004-3702(92)90071-5.

[12] Laura Giordano, Daniele Theseider Dupré, Camillo Fiorentini, Alberto Momigliano & Alberto Pettorossi (2018): *Defeasible Reasoning in $\mathcal{SROEL}$: from Rational Entailment to Rational Closure*. *Fundamenta Informaticae, Volume 161, Issue 1-2* 161(1–2), p. 135–161, doi:https://doi.org/10.3233/FI-2018-1698.

[13] Volker Halbach (19 August 2010): *The Logic Manual*. Oxford University Press.

[14] Frank van Harmelen, Frank van Harmelen, Vladimir Lifschitz & Bruce Porter (2007): *Handbook of Knowledge Representation*. Elsevier Science, San Diego, CA, USA.

[15] Adam Kaliski (2020): *An Overview of KLM-Style Defeasible Entailment. Master's thesis.* Ph.D. thesis, Faculty of Science, University of Cape Town, Rondebosch, Cape Town, 7700. Available at http://hdl.handle.net/11427/32743.

[16] Sarit Kraus, Daniel Lehmann & Menachem Magidor (2002): *Nonmonotonic Reasoning, Preferential Models and Cumulative Logics*. *CoRR* cs.AI/0202021, doi:https://doi.org/10.1016/0004-3702(90)90101-5.

[17] Daniel Lehmann (1995): *Another perspective on default reasoning*. *Annals of Mathematics and Artificial Intelligence* 15(1), pp. 61–82, doi:https://doi.org/10.1007/BF01535841.

[18] Daniel Lehmann & Menachem Magidor (May 1992): *What does a conditional knowledge base entail?* *Journal of Artificial Intelligence* Vol. 55 no.1, pp. 1–60, doi:https://doi.org/10.1016/0004-3702(92)90041-U.

[19] Vladimir Lifschitz (2002): *Answer Set Programming and Plan Generation*. *Artificial Intelligence* 138, pp. 39–54. Available at http://www.cs.utexas.edu/users/ai-lab?lif02.

[20] Vladimir Lifschitz (2008): *Twelve Definitions of a Stable Model*. *Logic Programming. ICLP 2008. Lecture Notes in Computer Science, vol 5366. Springer, Berlin, Heidelberg*, pp. 37–51, doi:https://doi.org/10.1007/978-3-540-89982-2_8.

[21] Vladimir Lifschitz (2019): *Answer Set Programming*. Springer Cham, doi:https://doi.org/10.1007/978-3-030-24658-7.

[22] Drew McDermott & Jon Doyle (1980): *Non-monotonic logic I*. *Artificial Intelligence* 13(1), pp. 41–72, doi:https://doi.org/10.1016/0004-3702(80)90012-0. Special Issue on Non-Monotonic Logic.

[23] Ben-Ari Mordechai (2012): *Mathematical logic for computer science*, 3 edition. Springer, London, doi:https://doi.org/10.1007/978-1-4471-4129-7.

[24] C.E. Alchourr ón, P. Gä rdenfors & D Makinson (1985): *On the Logic of Theory Change: Partial Meet Contraction and Revision Functions*. *J. Symb. Log.* 50, pp. 510–530, doi:https://doi.org/10.1007/978-3-319-20451-2_13.

[25] Stuart Russell & Peter Norvig (2021): *Artificial Intelligence, Global Edition A Modern Approach*. Pearson Deutschland. Available at https://elibrary.pearson.de/book/99.150005/9781292401171.

[26] Karl Schlechta (2007): *Nonmonotonic Logics: a Preferential Approach*. In Dov M. Gabbay & John Woods, editors: *The Many Valued and Nonmonotonic Turn in Logic*, *Handbook of the History of Logic* 8, North-Holland, pp. 451–516, doi:https://doi.org/10.1016/S1874-5857(07)80010-0.

[27] Yoav Shoham (1987): *Nonmonotonic Logics: Meaning and Utility*. *In Proceedings of the 10th International Joint Conference on Artificial Intelligence - Volume 1 (Milan, Italy) (IJCAI'87). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 388–393.*

A Appendix

A.1 Natural deduction system

A deductive system allows for the construction of proofs and the step-by-step establishment of tautologies. The system we use in our ASP encodings is natural deduction. The natural deduction system allows for the proof of conclusions via intermediate conclusions from a certain set of premises.

The natural deduction system is both sound and complete [13]. Formally, if Γ is defined as a set of premises and ω is the sentence that is being proved, then it is possible to pass from the set of premises Γ to ω iff $\Gamma \models \omega$. Furthermore, this system is complete in that all valid arguments ω, where $\Gamma \models \omega$, are reachable by going through the proof steps defined within this system.

Proofs in natural deduction start with an assumption. To conclude that $A \to B$ holds, we will start with the assumption of A and then conclude $A \to B$ from this.

In natural deduction, there are two important concepts when it comes to working with each connective: the introduction rule and the elimination rule. The introduction rule specifies how we introduce a connective to a proof, while the elimination rule specifies how we can eliminate connectives from proofs.

Proof of Transitivity

Given premises: $A \to B, B \to C$.

$$\dfrac{\dfrac{A, A \to B}{B} \quad B \to C}{\dfrac{C}{A \to C}}$$

Discharging the assumption, we obtain $A \to C$.

Proof of Contrapositive

Given premise: $A \to B$.

$$\dfrac{\dfrac{\neg B, \dfrac{A, A \to B}{B} \quad (\to E)}{\bot \quad (\neg E)}}{\neg A \quad (\neg I)} \neg B \to \neg A \quad (\to I)$$

Proof of Disjunction Elimination

Given premises: $A \vee \neg A, A \to C, \neg A \to C$.

$$\dfrac{A \vee \neg A, \quad \dfrac{A, A \to C}{C}, \quad \dfrac{\neg A, \neg A \to C}{C}}{C}$$

Proof of Identity

Given premise: $X \to Y$.

$$\dfrac{X \to Y}{X \to Y}$$

A Proofs for Section 3 (Declarative approach to BaseRank and Rational Closure)

Proposition 1. Let i be a fixed nonnegative integer and $\mathcal{S}$ be some answer set satisfying the rules outlined in `BaseRank` without the minimize directive. Further define

$$\mathrm{ND}_N \;=\; \left\{ X \to Y \;\middle|\; \overrightarrow{\mathcal{K}}_i^{\mathcal{S}} \models X \to Y \right\}$$

and

$$\mathrm{ASP}_i \;=\; \left\{ X \to Y \;\middle|\; \texttt{derive(X,Y,i)} \in \mathcal{S} \right\}.$$

Then

$$\mathrm{ND}_i \;=\; \mathrm{ASP}_i.$$

Proof. First note that, as our language is nothing but "if–then" between literals, the only non-trivial reasoning steps are identity, modus ponens, and disjunctive elimination.

We proceed by showing the following:

$$\mathrm{ND}_N \subseteq \mathrm{ASP}_N \quad \text{and} \quad \mathrm{ASP}_N \subseteq \mathrm{ND}_N.$$

(1) $\mathrm{ND}_N \subseteq \mathrm{ASP}_N$ Let

$$X \;\to\; Y \;\in\; \mathrm{ND}_N.$$

By definition, there is a natural deduction proof of $X \to Y$ from premises in $\mathcal{K}_N$, We refer to this proof as Π, which has length n. We show that $\texttt{derive}(X,Y,N) \in$ ASP_N. We proceed by structural induction on the length of deductive proof Π.

Base Case (n=0): If n=0, then this is simply the identity rule and $X \to Y$ is a premise in $\mathcal{K}_N$, from rule

```
derive(X,Y,N) :- rank(m_implication(X,Y),N) , derive(X,X,N).
```

is added into the answer set, this $\texttt{derive(X,Y,N)} \in \mathcal{S}$, $X \to Y \in ASP_N$.
Inductive Hypothesis (n=k): For any natural deductive proof Π of length k if $X \to Y \in ND_N$ then $X \to Y \in ASP_N$

Inductive Step ($n = k+1$). Let Π be a natural-deduction proof of $X \to Y$ of length $k+1$. We consider the following cases:

Idenity : In this case, the last step is "reiterate the premise $X \to Y$." In natural deduction notation:

$$\frac{}{X \to Y \;\vdash\; X \to Y} \quad\Longrightarrow\quad X \to Y.$$

Since $X \to Y \in \mathcal{K}_N$, the corresponding ASP clause

$$\texttt{derive}(X,Y,N) \;:-\; \texttt{rank}\big(\texttt{m_implication}(X,Y),N\big)$$

fires, so $\texttt{derive}(X,Y,N) \in \mathcal{S}$. Hence $X \to Y \in \mathrm{ASP}_N$.

Modus Ponens | Transitivity :

$$\frac{X \to Z \quad Z \to Y}{X \to Y}$$

By inductive hypothesis, both $X \to Z$ and $Z \to Y$ yield $\texttt{derive}(X,Z,N) \in \mathcal{S}$ and $\texttt{derive}(Z,Y,N) \in \mathcal{S}$. Then the ASP rule

$$\texttt{derive}(X,Y,N) \text{ :- } \texttt{derive}(X,Z,N), \texttt{derive}(Z,Y,N)$$

applies, giving $\texttt{derive}(X,Y,N)$. Thus $X \to Y \in \text{ASP}_N$.

Disjunction Elimination :

$$\frac{[X] \vdash Y \quad [\neg X] \vdash Y}{Y}$$

By the inductive hypothesis, we have $\texttt{derive}(X,Y,N) \in \mathcal{S}$ and $\texttt{derive}(-X,Y,N) \in \mathcal{S}$. Then ASP's disjunction-elimination clause

$$\texttt{derive}(\top,Y,N) \text{ :- } \texttt{derive}(X,Y,N), \texttt{derive}(-X,Y,N)$$

adding $\texttt{derive}(\top,Y,N)$ into the answer set, i.e. $Y \in \text{ASP}_N$.

In every case, $\texttt{derive}(X,Y,N)$ (or $\texttt{derive}(\top,Y,N)$) appears in $\mathcal{S}$. Hence $\text{ND}_N \subseteq \text{ASP}_N$.

(2) $\text{ASP}_N \subseteq \text{ND}_N$**.**

Conversely, suppose $\texttt{derive}(X,Y,N) \in \mathcal{S}$. By inspecting the BaseRank encoding, there are exactly three ways $\texttt{derive}(X,Y,N)$ can be derived:

1. $\texttt{derive}(\texttt{X},\texttt{Y},\texttt{N})$ is directly derived from $\texttt{rank}(\texttt{m_implications}(\texttt{X},\texttt{Y}),\texttt{N})$, line 10 from Listing 2.
2. $\texttt{derive}(\texttt{X},\texttt{Y},\texttt{N})$ is derived from $\texttt{derive}(\texttt{X},\texttt{Z},\texttt{N})$ and $\texttt{derive}(\texttt{Z},\texttt{Y},\texttt{N})$, line 11 from Listing 2.
3. $\texttt{derive}(\texttt{top},\texttt{Z},\texttt{N})$ is derived from $\texttt{derive}(\texttt{X},\texttt{Z},\texttt{N})$ and $\texttt{derive}(-\texttt{X},\texttt{Z},\texttt{N})$, line 12 from Listing 2.

We reconstruct a natural deduction proof of $X \to Y$ by induction on the ASP-derivation depth of the derive predicate, $\texttt{derive}(\texttt{X},\texttt{Y},\texttt{N})$.
Base Case (n=0): If the length of the derive predicate is 0 then $\texttt{derive}(X,Y,N)$ is added into the answer set $\mathcal{S}$ from $\texttt{rank}(\texttt{m_implication}(X,Y),N)$, thus$X \to Y \in \mathcal{K}_N$ and $X \to Y \in ND_N$.
Inductive hypothesis (n=k): For length n=k, then for every $\texttt{derive}(\texttt{X},\texttt{Y},\texttt{N}) \in \mathcal{S}$, for every $X \to Y \in ASP_N$, $X \to Y \in ND_N$.
Inductive Step (n=k+1):

- If $\texttt{derive}(\texttt{X},\texttt{Y},\texttt{N})$ was obtained from $\texttt{rank}(\texttt{m_implications}(\texttt{X},\texttt{Y}),\texttt{N})$ then this is simply the base case so this $X \to Y \in ND_N$.
- If $\texttt{derive}(X,Y,N)$ results from $\texttt{derive}(X,Z,N)$ together with $\texttt{rank}(\texttt{m_implication}(Z,Y),N)$. From the inductive hypothesis, $X \to Z$ and $Z \to Y \in ND_N$, thus through the use of natural deduction, we can show that $X \to Y \in ND_N$.
- If $\texttt{derive}(X,Y,N)$ and $\texttt{derive}(-X,Y,N)$ hold then $\texttt{derive}(\top,Y,N) \in \mathcal{S}$, by the inductive hypothesis, $X \to Y$ and $X \to \neg Y \in ND_N$, through natural deduction $\top \to Y \in ND_N$.

In each scenario, we build a valid ND subproof for $X \to Y$. Hence $\text{ASP}_N \subseteq \text{ND}_N$.

Combining (1) and (2) completes the proof that $\text{ND}_N = \text{ASP}_N$. □

Proposition 2. Let a knowledge base $\mathcal{K}$ be given. For a given answer set $\mathcal{S}$ of $\texttt{encoding}(\mathcal{K}) \cup \texttt{baseRank}$, for any $i \geq 0$, and for any $A \mid\sim B \in \mathcal{K}_i^{\mathcal{S}}$, $\bigcup_{j=i}^{\infty} \overrightarrow{\mathcal{K}_j^{\mathcal{S}}} \not\models \neg A$.

Proof. Assume by contradiction that for some $A \mid\sim B \in \mathcal{K}_i^{\mathcal{S}}$, $\bigcup_{j=i}^{\infty} \overrightarrow{\mathcal{K}_j^{\mathcal{S}}} \models \neg A$. This entails that every interpretation that satisfies $\bigcup_{j=i}^{\infty} \overrightarrow{\mathcal{K}_j^{\mathcal{S}}}$ also satisfies $\neg A$. From the equivalence between the ASP rules and natural deductive rules from 1 we see that this means $\texttt{derive}(\texttt{top}, -\texttt{A}, \texttt{j})$ is added into the answer set S. From this we see that using natural deduction rules, $\bigcup_{j=i}^{\infty} \overrightarrow{\mathcal{K}_j^{\mathcal{S}}} \models (A \rightarrow \bot)$. holds. However, given that $A \mid\sim B \in \mathcal{K}_i^{\mathcal{S}}$, this means $\texttt{derive}(\texttt{A},\texttt{B},\texttt{j}) \in \mathcal{S}$. From this we see that $\texttt{derive}(\texttt{A},\texttt{B},\texttt{j}), \texttt{derive}(\texttt{top}, -\texttt{A}, \texttt{j}) \in \mathcal{S}$. This does not hold due to the integrity constraint on line 15 of Listing 2. Thus $\bigcup_{j=i}^{\infty} \overrightarrow{\mathcal{K}_j^{\mathcal{S}}} \not\models \neg A$ □

Proposition 3. Let a knowledge base $\mathcal{K}$ with $\texttt{BaseRank}(\mathcal{K}) = (\mathcal{R}_0, \ldots, \mathcal{R}_n, \mathcal{R}_\infty)$ be given. For any partition $(\mathcal{R}'_0, \ldots, \mathcal{R}'_m, \mathcal{R}_\infty)$ of $\mathcal{K}$ s.t. for every $\mathcal{R}'_i$ there is a j s.t. $\mathcal{R}'_i \subseteq \mathcal{R}_j$, there is an answer set $\mathcal{S}$ of $\texttt{encoding}(\mathcal{K}) \cup \texttt{baseRank}$ s.t. $\mathcal{K}_i^{\mathcal{S}} = \mathcal{R}'_i$ for every $i \geq 0$.

Proof. Given a refinement $(R'_0, \ldots, R'_m, R_\infty)$ of the BaseRank partition $(R_0, \ldots, R_n, R_\infty)$, where for each R'_i there exists some j such that $R'_i \subseteq R_j$. Given R'_i, we can construct a answer set $\mathcal{S}$ as follows:

- For each defeasible implication $A \mid\sim B \in R'_i$, include the atom `rank(m_implication(A,B), i)` in S.
- For each classical implication $A \rightarrow B \in K$, include the atom `rank(m_implication(A,B), inf)` in S (as per the encoding).
- Include all atoms derivable from these rankings under the inference and propagation rules of the encoding, `derive/3`, `infer/2`.

Given this answer set $\mathcal{S}$ we will show that it is satisfied by the integrity constraints defined by `BaseRank`.

- **Integrity constraint on line 14**:
  This constraint forbids deriving $X \rightarrow Y$ and $X \rightarrow -Y$ simultaneously at rank N. The original BaseRank partition $(R_0, \ldots, R_n, R_\infty)$ is consistent by construction. Since every R'_i is a subset of some R_j, the refinement cannot introduce contradictions. Thus, line 18 holds.
- **Integrity constraint on lines 15-16**:
  These constraints forbid deriving both $-X$ when X is a tautology on the level N. Again, since the original partition is consistent, any subset is consistent. Thus, the integrity constraint holds.
- **Integrity constraint on lines 17**:
  The construction explicitly assigns exactly one rank to each implication within the knowledge base, $\mathcal{K}$, thus the integrity constraint holds.

Thus, $\mathcal{K}_i^S = \mathcal{R}'_i$ for each i. □

Proposition 4. Let a knowledge base $\mathcal{K}$ with $\mathit{BaseRank}(\mathcal{K}) = (\mathcal{R}_0, \ldots, \mathcal{R}_n, \mathcal{R}_\infty)$ be given. For the answer set $\mathcal{S}$ of $\texttt{encoding}(\mathcal{K}) \cup \texttt{baseRank} \cup \texttt{baseRankMin}$, $\mathcal{K}_i^S = \mathcal{R}_i$ for $i \geq 0$.

Proof. Suppose some defeasible conditional $\delta = \alpha \mid\sim \beta$ is placed at rank $j > i$ in the minimal answer set S, whereas $\delta \in R_i$ in $\mathit{BaseRank}(K)$. Since $\mathit{BaseRank}$ assigns δ to the lowest consistent rank, we know that placing δ at rank i would still yield a consistent interpretation.

Thus, we can construct an alternative answer set S' where δ is reassigned to rank i, keeping all other assignments unchanged. This new answer set would have a strictly lower cost under the *#minimize* directive, contradicting the assumption that S is minimal.

Conversely, suppose there exists a defeasible conditional $\delta \in R_i$ in $BaseRank(K)$ that does not appear at rank i in S. Then either δ is placed at a higher rank (contradicted above), or not ranked at all (which is forbidden by the choice rule). Hence, δ must appear in K_i^S.

It follows that the minimal answer set must assign each defeasible conditional to the same rank as in $BaseRank(K)$. Therefore, $\mathcal{K}_i^S = \mathcal{R}_i$ for all i. □

Proposition 5. Given some knowledge base $\mathcal{K}$ and query A$|\sim$ B, if $\texttt{entailed(true)} \in \mathcal{S}$, where $\mathcal{S}$ is the minimal answer set, then $\mathcal{K} \mid\approx_{RC}$ A $|\sim$ B.

Proof. Given $\texttt{entailed(true)} \in \mathcal{S}$, by the definition of entailed in RC this means $\texttt{inference(A,B)} \in \mathcal{S}$. To show that $\mathcal{K} \mid\approx_{RC}$ we will construct the corresponding BaseRank model using Π. Let $\bigcup_{j=n}^{\infty} \overrightarrow{\mathcal{K}_j^{\mathcal{S}}} = \{A \rightarrow B \mid \texttt{rank(m_implication(A,B),i)} \in \mathcal{S}, \texttt{guess(N)} \in \mathcal{S}\}$, where $n = N$ for $\texttt{guess(N)} \in \mathcal{S}$. By Proposition 3 $\bigcup_{j=n}^{\infty} \overrightarrow{\mathcal{K}_j^{\mathcal{S}}}$ maps onto a corresponding minimal base rank model used for RC, $\bigcup_{j=n}^{\infty} \overrightarrow{\mathcal{K}_j^{\mathcal{S}}} = \mathcal{R}_\infty \bigcup \mathcal{R}$. The inference rules mapped out in RC are exactly those used for modeling entailment, thus $\mathcal{R}_\infty \bigcup \mathcal{R} \vDash$ A $\rightarrow B$ leading to $\mathcal{K} \mid\approx_{RC} A \rightarrow B$ □

Proposition 6. Given some knowledge base $\mathcal{K}$ and query A$|\sim$ B, if $\mathcal{K} \mid\approx_{RC}$ A $|\sim$ B then there exists a minimal answer set $\mathcal{S}$ of Π where $\texttt{entailed(true)} \in \mathcal{S}$.

Proof. $\mathcal{K} \mid\approx_{RC} A \mid\sim B$, thus there exists some partition of the BaseRank model, $\mathcal{R}'$ where $\mathcal{R}_\infty \bigcup \mathcal{R}' \vDash$ A $\rightarrow B$. Proposition 3 ensures a corresponding answer set $\mathcal{S}$ which respects this ranking. Using the inference rules defined by the ASP encoding, it will then be derived that $\texttt{inference(A,B)} \in \mathcal{S}$ with $\texttt{query(A,B)} \in \mathcal{S}$ resulting in $\texttt{entailed(true)} \in \mathcal{S}$. This holds as the inference rules defined follow the natural deduction style derivation rules, which are sound and complete. □

Extracting Policies from Quantified Answer Set Programs

Martín Diéguez
University of Angers, France
martin.dieguezlodeiro@univ-angers.fr

Igor Stéphan
University of Angers, France
igor.stephan@univ-angers.fr

Quantified Answer Set Programming (QASP) extends Answer Set Programming (ASP) by allowing quantification over propositional variables, similar to Quantified Boolean Formulas (QBF). In this paper, we interpret models of QASP formulas in terms of *policies*, which represent decision-making strategies that determine how existentially quantified variables should be assigned, given the conditions set by universally quantified variables. As a main contribution, we present an algorithm for policy extraction under QASP semantics, inspired by the Equilibrium Logic semantics for general ASP theories.

1 Introduction

Having its roots in logic programming, *Answer Set Programming* [7] (ASP) is a formalism for nonmonotonic reasoning (NMR) and Knowledge Representation (KR) that solves complex problems involving combinatorial search, optimisation, and reasoning under uncertainty. In ASP, problems are encoded as a set of logical rules, where the corresponding models (answer sets) represent solutions. Besides the traditional *stable models* semantics, there exist alternative interpretations of logic programs under answer set semantics [15], including Pearce's *Equilibrium Logic* [18] (EL), which uses a monotonic basis and a minimality condition to induce nonmonotonicity. EL has been extended to enhance expressiveness for problems like temporal or epistemic reasoning. For propositional theories, the satisfiability problem in ASP lies within the second level of the polynomial hierarchy (Σ_2^p) [19]. Approaches like stable-unstable semantics [6] extend ASP to solve problems beyond this level by integrating logic programs as oracles.

A second approach to problem solving is the ASP(Q) system[1] [2], which extends the ASP syntax with existential and universal quantifiers that range over the stable models of a program rather than over atoms. In terms of applications, ASP(Q) has been employed in representing and solving graph problems [1], as well as in epistemic logic programming [10] and argumentation [9].

In the classical setting, *Quantified Boolean Formulas* [5, Chapter 31] (QBFs) extend propositional logic with second-order quantification, allowing for the expression of problems beyond the complexity class Σ_2^p, since QBF satisfiability is PSPACE-complete [22]. *Quantified Answer Set Programming* (QASP) [20, 11] similarly extends ASP by incorporating propositional quantifiers, analogous to QBF. As in QBF, both existential and universal quantifiers apply to the truth values of propositional atoms. However, the key difference lies in the semantics: in QASP, the truth values assigned to atoms must be consistent with the stable models of the program. There exist two primary semantics for QASP programs. In the semantics proposed by Fandinno et al. [11], an atom p is forced to be true in a stable model by adding the formula $\neg\neg p$ to the context. In contrast, in the semantics defined by Stephan [20], p is made true simply by adding p to the context. Both semantics behave identically when determining whether an atom is false in a stable model. In terms of implementation, the semantics from [11] are realized in a tool called

[1] Available at https://www.mat.unical.it/ricca/downloads/qasp-0.1.2.jar

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 224–237, doi:10.4204/EPTCS.439.16

`qasp2qbf`[2], which has been used to solve problems in conformant planning. Furthermore, it is shown in [11, Appendix B] that ASP(Q) and QASP are intertranslatable.

In QASP, solutions to quantified logic programs can be interpreted as two-player games with complete information. The first player attempts to construct stable models of the ASP program, and the program is considered a positive instance of the QASP problem if and only if a winning strategy exists for the first player. The tool `qasp2qbf` reduces a QASP program to a QBF, which is then passed to a QBF solver to determine satisfiability. If the QBF is satisfiable, the solver provides an assignment for the initial block of existential quantifiers. This means that for QASP programs beginning with a universal quantifier, the solver can only report satisfiability status, without producing a concrete strategy. At present, the tool does not support the construction of such two-player games. In contrast, the approach in [20] replaces existential quantifiers with Skolem functions [4], effectively simulating the existential player's choices. This method is capable of computing all possible two-player games that satisfy the QASP program.

In this paper, we examine [11] and [20] semantics from the perspective of Equilibrium Logic (EL). We introduce the following restrictions: we do not consider a specific logic programming language; instead, we translate any logic programming rule $H \leftarrow B$ into the implication $B \rightarrow H$ in propositional logic. Default negation (`not`) is replaced by the negation symbol ($\neg$) in propositional logic, and propositional formulas can be arbitrarily nested. Additionally, we treat any arbitrary propositional formula[3] as the matrix of a Quantified Boolean Formula (QBF).

The first contribution of this paper is to demonstrate that the two semantics are not equivalent. The second contribution is the extension of the notion of policy from the classical case to the quantified propositional Gödel logic QG_3 [3], which provides a monotonic basis for selecting minimal policies. The third contribution is the development of an algorithm that takes a QBF as input and computes the set of policies satisfying it under the semantics of [11]. This algorithm combines QBF and QG_3-policies during execution.

This paper is organised as follows: Section 2.1 introduces QBFs and the concept of policy. Section 2.2 formulates EL in terms of the Gödel logic G_3 plus a minimality condition, proving several interesting properties. Section 2.3 presents the two QASP semantics used in this paper. Section 3 introduces the semantics for propositional quantifiers in G_3 and the concept of QG_3-policy. In Section 4, we present an algorithm, based on EL, for extracting policies from QBFs under QASP semantics. The paper concludes with Section 5, discussing future work.

2 Background

2.1 QBFs and QBFs Policies

Let `VAR` be a (nonempty) set of atoms. A QBF (in *prenex normal form*) is generated by the grammar:

$$\varphi, \psi \quad ::= \quad p \mid \bot \mid (\varphi \wedge \psi) \mid (\varphi \vee \psi) \mid (\varphi \rightarrow \psi) \tag{1}$$

$$F \quad ::= \quad \varphi \mid \forall x\, F \mid \exists x\, F, \tag{2}$$

where $p, x \in$ `VAR`. We will usually use the common rules for the elimination of parentheses, when necessary. The first part of the grammar (1) generates propositional formulas while (2) generates the prefix of quantifiers. Given a QBF $Q_1 x_1 \cdots Q_n x_n \varphi$ where $Q_i \in \{\forall, \exists\}$ and $x_i \in$ `VAR`, for all $1 \leq i \leq n$ as $\overline{Q}\varphi$, we will

[2] Available at `https://github.com/potassco/qasp2qbf`.

[3] Since EL allows interpreting arbitrary (propositional) formulas.

name the sequence $Q_1x_1 \cdots Q_nx_n$ the *binder* and we will name the quantifier-free formula φ the *matrix*. Symbol ε denotes the empty binder. For the semantics we define a *classical interpretation* as a mapping $\boldsymbol{m} : \mathtt{VAR} \mapsto \{0,1\}$. For a given propositional variable p, $\boldsymbol{m}(p) = 0$ means that p is false; $\boldsymbol{m}(p) = 1$ means that p is true. An interpretation $\boldsymbol{m}$ is said to be *total* if it is defined for all $p \in \mathtt{VAR}$ and *partial* otherwise. If not specified explicitly, we will assume that $\boldsymbol{m}$ is total. Given an interpretation $\boldsymbol{m}$, $x \in \mathtt{VAR}$ and $i \in \{0,1\}$, we define the *update* of x in $\boldsymbol{m}$ as

$$\boldsymbol{m}_i^x(p) \stackrel{\text{def}}{=} \begin{cases} i & \text{if } p = x \\ \boldsymbol{m}(p) & \text{otherwise.} \end{cases} \tag{3}$$

Moreover, by $\boldsymbol{\varepsilon}$ we denote the (partial) interpretation such that for all $x \in \mathtt{VAR}$, $\boldsymbol{\varepsilon}(x)$ is not defined. Any interpretation $\boldsymbol{m}$ can be extended to any formula φ by means of the following satisfaction relation:

$$\boldsymbol{m}(\bot) = 0 \qquad \boldsymbol{m}(\varphi \wedge \psi) = \min\{\boldsymbol{m}(\varphi), \boldsymbol{m}(\psi)\} \qquad \boldsymbol{m}(\varphi \vee \psi) = \max\{\boldsymbol{m}(\varphi), \boldsymbol{m}(\psi)\}$$
$$\boldsymbol{m}(\varphi \to \psi) = \begin{cases} 1 & \text{if } \boldsymbol{m}(\varphi) \leq \boldsymbol{m}(\psi) \\ 0 & \text{otherwise.} \end{cases}$$

For a theory Γ, we define $\boldsymbol{m}(\Gamma) = \min\{\boldsymbol{m}(\varphi) \mid \varphi \in \Gamma\}$. We say that an interpretation $\boldsymbol{m}$ is a *model* of a formula φ if $\boldsymbol{m}(\varphi) = 1$.

When a QBF is given in prenex normal form, semantics can be given in terms of QBF-policies. A *QBF-policy* (or policy in [8]) refers to a strategy or decision rule that dictates how to assign truth values to the variables in a QBF to achieve a desired outcome. Since QBF extends classical logic with second order quantification over the propositional variables, a policy can represent a systematic approach for deciding the truth values of variables depending on the quantifier structure.

In the context of Boolean logic, evaluating a propositional variable p to 1 (resp. 0) on a formula φ is equivalent to replacing all occurrences of p by $\top$ (resp. by $\bot$). Unfortunately, the same approach cannot be represented within the context of many-valued logics [16], where there are truth values that are not captured with elements of the language. Our definition of QBF-policies can be adapted easily to the case of many-valued logics, as done in Section 3.

Definition 1 (QBF-policy [8])**.** *The set $TP_{QBF}(\overline{Q})$ of QBF-policies for a binder $\overline{Q}$ is defined, recursively, as follows:*

$$\begin{aligned} TP_{QBF}(\varepsilon) &= \{\lambda\} \\ TP_{QBF}(\exists x_i \dots Q_n x_n) &= \{v; \pi_{i+1} \mid v \in \{0,1\}, \pi_{i+1} \in TP_{QBF}(Q_{i+1}x_{i+1} \dots Q_n x_n)\} \\ TP_{QBF}(\forall x_i \dots Q_n x_n) &= \{[0 \to \pi_{i+1}^0 \mid 1 \to \pi_{i+1}^1] \mid \\ &\qquad \pi_{i+1}^0, \pi_{i+1}^1 \in TP_{QBF}(Q_{i+1}x_{i+1} \dots Q_n x_n)\} \end{aligned}$$

where λ represents the empty QBF-policy. *The operator ";" represents the sequential composition of QBF-policies and $\{0,1\} \to \pi_{i+1}$ represents all possible functions from assignments of x_i to values of $\{0,1\}$ to QBF-policies in $TP_{QBF}(Q_{i+1}x_{i+1} \dots Q_n x_n)$.*

The satisfaction of a QBF is based on the concept of configuration. A *configuration* is the structure $\langle \boldsymbol{m}, \pi \rangle$, where $\boldsymbol{m}$ is a partial classical interpretation used to store the different assignments obtained during the analysis of the QBF-policy π. Given a QBF $Q_1x_1, \cdots, Q_nx_n\varphi$ and a configuration $\langle \boldsymbol{m}, \pi \rangle$, the satisfaction relation is defined, by cases, as

- $\langle \boldsymbol{m}, \lambda \rangle \models \varphi$ if $\boldsymbol{m}(\varphi) = 1$

- $\langle \boldsymbol{m}, \pi \rangle \models \exists x Q_{i+1} x_{i+1} \cdots Q_n\, x_n \varphi$ if for some $v \in \{0,1\}$, $\pi = v;\pi'$ and $\langle \boldsymbol{m}^x_v, \pi' \rangle \models Q_{i+1} x_{i+1} \cdots Q_n\, x_n \varphi$.
- $\langle \boldsymbol{m}, \pi \rangle \models \forall x Q_{i+1} x_{i+1} \cdots Q_n\, x_n \varphi$ if $\pi = [0 \to \pi^0 \mid 1 \to \pi^1]$ and both $\langle \boldsymbol{m}^x_0, \pi^0 \rangle \models Q_{i+1} x_{i+1} \cdots Q_n\, x_n \varphi$ and $\langle \boldsymbol{m}^x_1, \pi^1 \rangle \models Q_{i+1} x_{i+1} \cdots Q_n\, x_n \varphi$.

Definition 2. *A QBF-policy π is said to satisfy a QBF $Q_1 x_1, \cdots, Q_n x_n \varphi$ if $\langle \boldsymbol{\varepsilon}, \pi \rangle \models Q_1 x_1, \cdots, Q_n x_n \varphi$.*

Example 1. *Let us consider the QBF $\varphi = \forall x\, \exists z\, ((z \to x) \wedge (z \vee \neg z))$. There exist two QBF-policies that satisfy φ: $\pi_{(a)} = [0 \to 0; \lambda \mid 1 \to 0; \lambda]$ and $\pi_{(b)} = [0 \to 0; \lambda \mid 1 \to 1; \lambda]$ with a tree-shape representation shown in Figure 1:*

Figure 1: Two policies for the QBF $\psi = \forall x \exists z\, ((z \to x) \wedge (z \vee \neg z))$.

Note that, since $z \vee \neg z$ is a tautology, φ is equivalent to $\varphi = \forall x\, \exists z (z \to x)$. Therefore, it is easy to see that, if x is evaluated to 0, we are forced to evaluate z to 0. On the contrary, when x is evaluated to 1, the quantified formula is satisfied regardless the truth value of z.

2.2 Propositional Equilibrium Logic

EL shares its syntax with propositional logic. Therefore, formulas are generated by means of the grammar presented in (1). The semantics of EL are defined in terms of the (monotonic) logic of *Here-and-There* [14, 12] (HT) together with a minimality criterion that induces the non-monotonicity. However, for our purposes we present an equivalent definition in terms of the three-valued Gödel logic G_3 [12]. Within this logic, an interpretation is defined as a three-valued mapping $\boldsymbol{m} : \mathtt{VAR} \mapsto \{0, \frac{1}{2}, 1\}$. As in classical logic, given $p \in \mathtt{VAR}$, $\boldsymbol{m}(p) = 0$ means that p is false; $\boldsymbol{m}(p) = 1$ means that p is true. In addition, $\boldsymbol{m}(p) = \frac{1}{2}$ means that p is true by default. Given an interpretation $\boldsymbol{m}$, $x \in \mathtt{VAR}$ and $i \in \{0, \frac{1}{2}, 1\}$, the concept of *update* is defined now as in (3) with the peculiarity that now $i \in \{0, \frac{1}{2}, 1\}$. The semantics correspond to the propositional case except for the implication $\varphi \to \psi$, which is interpreted as

- $\boldsymbol{m}(\varphi \to \psi) = \begin{cases} 1 & \text{if } \boldsymbol{m}(\varphi) \leq \boldsymbol{m}(\psi) \\ \boldsymbol{m}(\psi) & \text{otherwise.} \end{cases}$

Note that, in G_3 (so in classical logic as well), $\neg\varphi \stackrel{\text{def}}{=} \varphi \to \bot$. As in the classical case, we say that a G_3 interpretation $\boldsymbol{m}$ is a *model* of a formula φ if $\boldsymbol{m}(\varphi) = 1$.

Proposition 1 ([18])**.** *Here-and-There logic and the Gödel G_3 are equivalent.*

Given an interpretation $\boldsymbol{m}$ and $\Sigma \subseteq \mathtt{VAR}$, we define the interpretation $\mathtt{crisp}^{\boldsymbol{m}}_{\Sigma}$ [4] as $\mathtt{crisp}^{\boldsymbol{m}}_{\Sigma}(p) = 1$ if $\boldsymbol{m}(p) = \frac{1}{2}$ and $p \in \Sigma$; $\mathtt{crisp}^{\boldsymbol{m}}_{\Sigma}(p) = \boldsymbol{m}(p)$, otherwise. For simplicity, we will denote $\mathtt{crisp}^{\boldsymbol{m}} \stackrel{\text{def}}{=} \mathtt{crisp}^{\boldsymbol{m}}_{\mathtt{VAR}}$.

Proposition 2. *For all interpretations $\boldsymbol{m}$ and formulas φ, $\mathtt{crisp}^{\boldsymbol{m}}(\varphi) \in \{0,1\}$.*

[4] The term crisp has been coined in the area of fuzzy logics to denote "classical interpretations"

Proposition 3 (persistency)**.** *For all formulas φ and for all interpretations $\boldsymbol{m}$, (1) $\boldsymbol{m}(\varphi) \neq 0$ implies $\mathtt{crisp}^{\boldsymbol{m}}(\varphi) = 1$ and (2) $\boldsymbol{m}(\varphi) = 0$ implies $\mathtt{crisp}^{\boldsymbol{m}}(\varphi) = 0$.*

Corollary 4. *For all formulas φ and interpretations $\boldsymbol{m}$, $\boldsymbol{m}(\neg\varphi) = 1$ iff $\mathtt{crisp}^{\boldsymbol{m}}(\varphi) = 0$.*

Proposition 2 states that crisp interpretations can be regarded as classical interpretations. We can relate two interpretations $\boldsymbol{m}$ and $\boldsymbol{m'}$, with respect to $\Sigma \subseteq \mathtt{VAR}$, by saying that

- $\boldsymbol{m} \triangleq_\Sigma \boldsymbol{m'}$ if $\boldsymbol{m}(p) = \boldsymbol{m'}(p)$, for all $p \in \Sigma$.
- $\boldsymbol{m} \trianglelefteq_\Sigma \boldsymbol{m'}$ if $\mathtt{crisp}^{\boldsymbol{m}}_\Sigma \triangleq_\Sigma \mathtt{crisp}^{\boldsymbol{m'}}_\Sigma$ and $\boldsymbol{m}(p) \leq \boldsymbol{m'}(p)$, for all $p \in \Sigma$.
- $\boldsymbol{m} \triangleleft_\Sigma \boldsymbol{m'}$ if $\boldsymbol{m} \trianglelefteq_\Sigma \boldsymbol{m'}$ and not $\boldsymbol{m} \triangleq_\Sigma \boldsymbol{m'}$.

When $\Sigma = \mathtt{VAR}$, the suffix Σ will be omitted from the context in the relations above.

Definition 3 (Equilibrium Model)**.** *We say that an interpretation $\boldsymbol{m}$ is an equilibrium model of a propositional formula φ if the following conditions hold:*

(1) $\boldsymbol{m} = \mathtt{crisp}^{\boldsymbol{m}}_{\mathtt{VAR}}$, i.e. $\boldsymbol{m}$ is a classical model.

(2) $\boldsymbol{m}$ is $\trianglelefteq_{\mathtt{VAR}}$-minimal, i.e. there is no $\boldsymbol{m'} \triangleleft_{\mathtt{VAR}} \boldsymbol{m}$ satisfying $\boldsymbol{m'}(\varphi) = 1$.

Example 2 (Example 1 continued)**.** *Let us consider the formula $\varphi = ((z \rightarrow x) \wedge (z \vee \neg z))$. This program corresponds to the logic program* `{ x :- z. {z}. }`*, which has two answer sets: $\emptyset$ and $\{x,z\}$. The three-valued truth table, containing all possible G_3-interpretations for φ is displayed below.*

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
x	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	1
z	0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1
$\boldsymbol{m_i}(\varphi)$	1	0	0	1	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	1

It can be checked that $\boldsymbol{m_1}$, $\boldsymbol{m_4}$, $\boldsymbol{m_7}$ and $\boldsymbol{m_9}$ are the models of φ. Moreover, $\boldsymbol{m_1}$, $\boldsymbol{m_7}$ and $\boldsymbol{m_9}$ are crisp so they are the candidates to become equilibrium models. Both $\boldsymbol{m_1}$ and $\boldsymbol{m_9}$ are $\trianglelefteq$-minimal so they are equilibrium models that correspond to the answer sets $\emptyset$ and $\{x,z\}$, respectively. On the contrary, it can be checked that $\boldsymbol{m_4} \triangleleft \boldsymbol{m_7}$, so $\boldsymbol{m_7}$ is not an equilibrium model. The following propositions show that formulas of the form $\neg x$ (resp. $\neg\neg x$), with $x \in \mathtt{VAR}$, can be used to select equilibrium models where x is false (resp. true).

Proposition 5. *Let Γ be a propositional theory and let x be a propositional variable. For all G_3-interpretations $\boldsymbol{m}$, $\boldsymbol{m}$ is an equilibrium model of $\Gamma \cup \{\neg\neg x\}$ iff $\boldsymbol{m}$ is an equilibrium model of Γ and $\boldsymbol{m}(x) = 1$.*

Proposition 6. *Let Γ be a propositional theory and let x be a propositional variable. For all G_3- interpretations $\boldsymbol{m}$, $\boldsymbol{m}$ is an equilibrium model of $\Gamma \cup \{\neg x\}$ iff $\boldsymbol{m}$ is an equilibrium model of Γ and $\boldsymbol{m}(x) = 0$.*

Finally, it is worth to remark that EL and answer sets semantics coincide.

Theorem 7 ([18])**.** *There is a bijection between equilibrium models and answer sets of any arbitrary propositional theory.*

2.3 Quantified Answer Set Programming

In this section we present the semantics published by [20] and in [11]. However, as a difference, we allow any propositional formula (or theory) as a matrix of the QBF. The semantics defined in [11] are oriented towards satisfiability within answer set semantics and they are defined next.

Definition 4 (from [11]). *A logic program P is said to be* satisfiable *if it has an answer set. The satisfiability of a QASP program $\overline{Q}\,P$ is defined as follows:*

1. *$\exists x\,P$ is satisfiable if either $P\cup\{\neg\neg x\}$ or $P\cup\{\neg x\}$ is satisfiable.*
2. *$\forall x\,P$ is satisfiable if both $P\cup\{\neg\neg x\}$ and $P\cup\{\neg x\}$ are satisfiable.*
3. *$\exists x\overline{Q}\,P$ is satisfiable if either $\overline{Q}(P\cup\{\neg\neg x\})$ or $\overline{Q}(P\cup\{\neg x\})$ is satisfiable.*
4. *$\forall x\overline{Q}\,P$ is satisfiable if both $\overline{Q}(P\cup\{\neg\neg x\})$ and $\overline{Q}(P\cup\{\neg x\})$ are satisfiable.*

The satisfiability of a quantified variable is tested by adding $\neg\neg x$ (resp. $\neg x$) to P for every quantified variable x. In view of Propositions 6 and 5, those formulas allow selecting equilibrium models of P where propositional variables occur either positively or negatively. Moreover, they prove the following complexity result.

Definition 5 (from [20]). *A logic program P is said to be* satisfiable *if it has an answer set. The satisfiability of a QASP program $\overline{Q}\,P$ is defined as follows:*

1. *$\exists x\,P$ is satisfiable if either $P\cup\{x\}$ or $P\cup\{\neg x\}$ is satisfiable.*
2. *$\forall x\,P$ is satisfiable if both $P\cup\{x\}$ and $P\cup\{\neg x\}$ are satisfiable.*
3. *$\exists x\overline{Q}\,P$ is satisfiable if either $\overline{Q}(P\cup\{x\})$ or $\overline{Q}(P\cup\{\neg x\})$ is satisfiable.*
4. *$\forall x\overline{Q}\,P$ is satisfiable if both $\overline{Q}(P\cup\{x\})$ and $\overline{Q}(P\cup\{\neg x\})$ are satisfiable.*

Stéphan's semantics differ from Fandinno et al. in the way variables are forced to be true within an answer set. In [20] a variable x is forced to become true by adding x directly to the context. Therefore, x becomes directly true. In [11] instead, x is forced to become true by adding $\neg\neg x$ to the context. In view of Proposition 5, any equilibrium model must satisfy x. Therefore, truth value of x must be derivable from program itself. The following proposition can be used to show that the semantics of Definition 4 are weaker that the ones of Definition 5.

Proposition 8. *For all theory Γ and for all $x\in$ VAR and for all interpretations $\boldsymbol{m}$, if $\boldsymbol{m}$ is an equilibrium model of $\Gamma\cup\{\neg\neg x\}$ then $\boldsymbol{m}$ is an equilibrium model of $\Gamma\cup\{x\}$.*

Corollary 9. *If a quantified theory $\overline{Q}\Gamma$ is satisfiable w.r.t. Definition 4 then it is satisfiable w.r.t. Definition 5.*

Lemma 10. *The semantics proposed in Definition 4 and Definition 5 are not equivalent.*

Proof. Let us consider the following quantified propositional theory:

$$\forall x\exists y\exists z\underbrace{\{z\to x,\neg z\to y,\neg y\to z\}}_{\Gamma} \tag{4}$$

The answer sets of Γ are $\{x,z\}$ and $\{y\}$. From the point of view of satisfiability, the following example is satisfiable with respect to both definitions: the reader can check that the policy of Figure 2a is accepted by the semantics of both Definition 4 and Definition 5. However, if we compute all possible policies associated to (4) we will see that the policy of Figure 2b is accepted by the semantics of Definition 5 but not by the semantics of Definition 4. Notice that the policy of Figure 2b would lead to the evaluation of the answer set $\{x,y\}$, which is not an answer set of Γ. Since the first quantifier in (4) is universal, the system `asp2qbf` only answers satisfiable but it does not provide more information allowing us to determine that the policy in Figure 2b is not generated by the semantics of Definition 4.

□

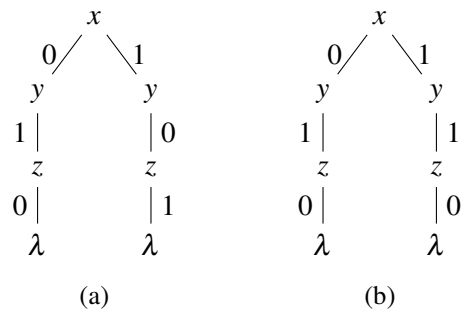


Figure 2: Two policies for the quantified theory (4).

3 QG_3 Policies

In the first place, we provide an interpretation of the second-order quantifiers within the context of G_3: given a G_3 interpretation $\boldsymbol{m}$, the semantics of quantifiers is defined as

$$\boldsymbol{m}(\forall x\varphi) = \min\left\{\boldsymbol{m}_i^x(\varphi) \mid i \in \{0,\frac{1}{2},1\}\right\} \qquad \boldsymbol{m}(\exists x\varphi) = \max\left\{\boldsymbol{m}_i^x(\varphi) \mid i \in \{0,\frac{1}{2},1\}\right\}.$$

The resulting logic is called *Quantified* G_3 (QG_3) and follows [3]. We extend the concept of QBF-policy to the case of QG_3.

Definition 6 (QG_3-policy)**.** *The set $TP_{QG3}(\overline{Q})$ of QG_3-policies for a binder $\overline{Q}$ is defined, recursively, as follows:*

$$\begin{aligned}
TP_{QG3}(\varepsilon) &= \{\lambda\} \\
TP_{QG3}(\exists x_i \ldots Q_n x_n) &= \{v;\pi_{i+1} \mid v \in \{0,\frac{1}{2},1\}, \pi_{i+1} \in TP_{QG3}(Q_{i+1}x_{i+1}\ldots Q_n x_n)\} \\
TP_{QG3}(\forall x_i \ldots Q_n x_n) &= \{[0 \to \pi_{i+1}^0 \mid \frac{1}{2} \to \pi_{i+1}^{\frac{1}{2}} \mid 1 \to \pi_{i+1}^1] \mid \\
&\qquad \pi_{i+1}^0, \pi_{i+1}^{\frac{1}{2}}, \pi_{i+1}^1 \in TP_{QG3}(Q_{i+1}x_{i+1}\ldots Q_n x_n)\}.
\end{aligned}$$

The operator ";" represents the sequential composition of QG_3-policies and $\{0,\frac{1}{2},1\} \to \pi_{i+1}$ refers to all possible functions from assignments of x_i to values of $\{0,\frac{1}{2},1\}$ to QG_3-policies in $TP_{QG3}(Q_{i+1}x_{i+1}\ldots Q_n x_n)$. A G_3 interpretation $\boldsymbol{m}$ is in a QG_3-policy π w.r.t. a binder $\overline{Q}$ ($\boldsymbol{m} \in_{\overline{Q}} \pi$ in symbols), if:

1) $\boldsymbol{m} \in_{\exists x\overline{Q}} \{v;\pi\}$ *if* $\boldsymbol{m}(x) = v$ *and* $\boldsymbol{m} \in_{\overline{Q}} \pi$
2) $\boldsymbol{m} \in_{\forall x\overline{Q}} [0 \to \pi^0 \mid \frac{1}{2} \to \pi^{\frac{1}{2}} \mid 1 \to \pi^1]$ *if* $\boldsymbol{m}(x) = v$ *and* $\boldsymbol{m} \in_{\overline{Q}} \pi^v$.

By extension, a G_3 interpretation $\boldsymbol{m}$ is in a configuration $\langle \boldsymbol{m}', \pi\rangle$ w.r.t. a binder $\overline{Q}$, $\boldsymbol{m} \in_{\overline{Q}} \langle \boldsymbol{m}', \pi\rangle$ in symbols, if $\boldsymbol{m} \in_{\overline{Q}} \pi$ and $\boldsymbol{m} \triangleq_{\mathtt{VAR}\setminus\mathtt{VAR}(\overline{Q})} \boldsymbol{m}'$.

In QG_3, a *configuration* is still a structure $\langle \boldsymbol{m}, \pi\rangle$, but now $\boldsymbol{m}$ is a partial G_3 interpretation used to store the different assignments obtained during the analysis of the QG_3-policy π. Given a QBF $Q_1x_1, \cdots, Q_nx_n\varphi$ and a configuration $\langle \boldsymbol{m}, \pi\rangle$, the satisfaction relation is defined as

- $\langle \boldsymbol{m}, \lambda\rangle \models \varphi$ if $\boldsymbol{m}(\varphi) = 1$
- $\langle \boldsymbol{m}, \pi\rangle \models \exists x Q_{i+1}x_{i+1}\cdots Q_n\, x_n\varphi$ if for some $v \in \{0,\frac{1}{2},1\}$, $\pi = v;\pi'$ and $\langle \boldsymbol{m}_v^x, \pi'\rangle \models Q_{i+1}x_{i+1}\cdots Q_n\, x_n\varphi$

- $\langle \boldsymbol{m}, \pi \rangle \models \forall x Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$ if $\pi = [0 \to \pi^0 \mid \frac{1}{2} \to \pi^{\frac{1}{2}} \mid 1 \to \pi^1]$ and $\langle \boldsymbol{m}^x_0, \pi^0 \rangle \models Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$, $\langle \boldsymbol{m}^x_{\frac{1}{2}}, \pi^{\frac{1}{2}} \rangle \models Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$ and $\langle \boldsymbol{m}^x_1, \pi^1 \rangle \models Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$.

Definition 7. *A QG_3-policy π is said to satisfy a QBF $Q_1 x_1 \cdots Q_n x_n \varphi$ if $\langle \boldsymbol{\varepsilon}, \pi \rangle \models Q_1 x_1 \cdots Q_n x_n \varphi$.*

Example 3 (Example 2 continued)**.** *Let us consider the formula $\psi = \forall x \exists z ((z \to x) \wedge (z \vee \neg z))$. The two QG_3-policies satisfying ψ are $\pi_{(a)} = [0 \to 0; \lambda \mid \frac{1}{2} \to 0; \lambda \mid 1 \to 0; \lambda]$ and $\pi_{(b)} = [0 \to 0; \lambda \mid \frac{1}{2} \to 0; \lambda \mid 1 \to 1; \lambda]$ and their tree-shape representations are shown in Figure 3.*

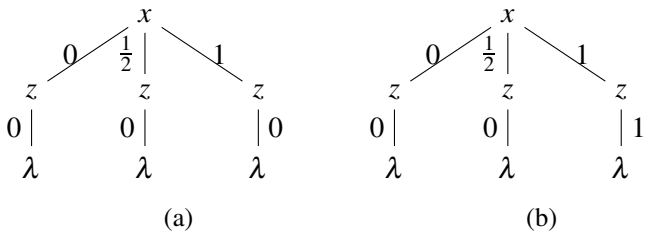


Figure 3: Two QG_3-policies satisfying the QBF $\psi = \forall x \exists z ((z \to x) \wedge (z \vee \neg z))$.

The following lemma shows that satisfiability according to QG_3-semantics of a QBF $\overline{Q}\varphi$ with no free variables (ie a variable that has an occurrence in φ but no one in $\overline{Q}$) depends only on the variables of φ.

Lemma 11. *Let $Q_1 x_1 \cdots Q_n x_n \varphi$ a QBF such that $\mathtt{VAR}(\varphi) = \{x_1, \ldots, x_n\}$ and $\boldsymbol{m}$ a G_3 interpretation. $\langle \boldsymbol{\varepsilon}, \pi \rangle \models Q_1 x_1 \cdots Q_n x_n \varphi$ if and only if $\langle \boldsymbol{m}, \pi \rangle \models Q_1 x_1 \cdots Q_n x_n \varphi$.*

The notion of minimal (or equilibrium) model for a propositional formula φ relies on the partial order $\trianglelefteq$ induced among the different interpretations. In the case of QBFs the truth values of the propositional variables are fixed by the interpretation of the quantifiers. As a consequence, its answer sets are represented in terms of *equilibrium configurations*, defined below.

Definition 8 (Equilibrium Configuration)**.** *Given a QBF $\overline{Q}\varphi$ and a configuration $\langle \boldsymbol{m}, \pi \rangle$, we define the notion of $\langle \boldsymbol{m}, \pi \rangle$ being an equilibrium configuration recursively as follows:*

- *If $\overline{Q}\varphi$ has no quantifiers then $\langle \boldsymbol{m}, \pi \rangle$ is an equilibrium configuration of $\overline{Q}\varphi$ if $\pi = \lambda$ and $\boldsymbol{m}$ is an equilibrium model of φ.*
- *If $\overline{Q}\varphi$ is of the form $\exists x Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$ then $\langle \boldsymbol{m}, \pi \rangle$ is an equilibrium configuration of $\overline{Q}\varphi$ if both 1) $\pi = v; \pi'$ with $v \in \{0,1\}$ and 2) $\langle \boldsymbol{m}^x_v, \pi' \rangle$ is an equilibrium configuration of $Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$*
- *If $\overline{Q}\varphi$ is of the form $\forall x Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$ then $\langle \boldsymbol{m}, \pi \rangle$ is an equilibrium configuration of $\overline{Q}\varphi$ if $\pi = [0 \to \pi^0 \mid 1 \to \pi^1]$ and both 1) $\langle \boldsymbol{m}^x_0, \pi^0 \rangle$ is an equilibrium configuration of $Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$ and 2) $\langle \boldsymbol{m}^x_1, \pi^1 \rangle$ is an equilibrium configuration of $Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$.*

A QBF-policy π is said to be an equilibrium policy of a quantified Boolean formula $\overline{Q}\varphi$ if there exists an equilibrium configuration $\langle \boldsymbol{m}, \pi \rangle$ of $\overline{Q}\varphi$.

4 Algorithm for Equilibrium Policies

We propose an algorithm that computes the QBF-policies of a QBF according to the equilibrium policies of the previous section. Our algorithm uses configurations $\langle \boldsymbol{m}, \pi \rangle$ where $\boldsymbol{m}$ is a G_3 interpretation and π is a QBF-policy instead of a G_3-policy in order to compute the equilibrium policies. To do this, we modify the satisfaction of QBFs as follows (the case $\frac{1}{2}$ is deleted from the semantics of the quantifier but not from the interpretation $\boldsymbol{m}$ from G_3):

- $\langle \boldsymbol{m}, \lambda \rangle \models \varphi$ if $\boldsymbol{m}(\varphi) = 1$
- $\langle \boldsymbol{m}, \pi \rangle \models \exists x Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$ if 1) $\pi = v; \pi'$, with $v \in \{0,1\}$ and 2) $\langle \boldsymbol{m}^x_v, \pi' \rangle \models Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$
- $\langle \boldsymbol{m}, \pi \rangle \models \forall x Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$ if $\pi = [0 \to \pi^0 \mid 1 \to \pi^1]$ and both 1) $\langle \boldsymbol{m}^x_0, \pi^0 \rangle \models Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$ and 2) $\langle \boldsymbol{m}^x_1, \pi^1 \rangle \models Q_{i+1} x_{i+1} \cdots Q_n x_n \varphi$.

Our algorithm realises the minimization thanks to a new structure, named Θ-pair, that captures some (potential) sub-policies plus the non-crisp models that may delete by minimisation some crisp models (and, by the same token, some potential policies):

Definition 9 ($\Theta^{\varphi}_{\overline{Q}}$-pair conditioned by an interpretation $\boldsymbol{m}$)**.** *Let $\overline{Q}$ be a binder and φ a quantifier-free formula. A $\Theta^{\varphi}_{\overline{Q}}$-pair conditioned by the interpretation $\boldsymbol{m}$ is a pair (T, HC) consisting of a set of QBF-policies $T \subset TP_{QBF}(\overline{Q})$ such that for all $\pi \in T$, $\langle \boldsymbol{m}, \pi \rangle \models \overline{Q}\varphi$, and a set HC of non-crisp models of φ such that for all $\boldsymbol{m}' \in HC, \boldsymbol{m}' \triangleq_{\mathtt{VAR}(\varphi) \setminus \mathtt{VAR}(\overline{Q})} \boldsymbol{m}$ and $\boldsymbol{m}' \not\triangleq_{\mathtt{VAR}(\overline{Q})} \boldsymbol{m}$.*

The following example illustrates the base cases $\overline{Q} = \exists z$ and $\overline{Q} = \forall z$ of our quantifier elimination algorithm for QASP.

Example 4 (Example 2 continued)**.** *Let $\varphi = ((z \to x) \wedge (z \vee \neg z))$ be the formula of Example 4. We consider a binder of the form $Qx\,Qz$ meaning that the variable x is eliminated before z. In that case, we assume that x has been (already) assigned with a value during the elimination process. If $Qz = \exists z$ we will reason as follows:*

- *Since $\langle \boldsymbol{m}_1, \lambda \rangle \models \varphi$ and $\boldsymbol{m}_2(\varphi) = \boldsymbol{m}_3(\varphi) = 0$ then $\langle \boldsymbol{\varepsilon}^x_0, 0; \lambda \rangle \models \exists z \varphi$ and $(T^{\exists z}_0, HC^{\exists z}_0) = (\{0; \lambda\}, \emptyset)$ is a $\Theta^{\varphi}_{\exists z}$-pair conditioned by $\boldsymbol{\varepsilon}^x_0$;*
- *Since $\boldsymbol{m}_5(\varphi) = \boldsymbol{m}_6(\varphi) = \frac{1}{2}$ and $\boldsymbol{m}_4(\varphi) = 1$ and $\boldsymbol{m}_4$ is a non-crisp model of φ then $(T^{\exists z}_{\frac{1}{2}}, HC^{\exists z}_{\frac{1}{2}}) = (\emptyset, \{\boldsymbol{m}_4\})$ is a $\Theta^{\varphi}_{\exists z}$-pair conditioned by $\boldsymbol{\varepsilon}^x_{\frac{1}{2}}$;*
- *Since $\langle \boldsymbol{m}_7, \lambda \rangle \models \varphi$ and $\langle \boldsymbol{m}_9, \lambda \rangle \models \varphi$ and $\boldsymbol{m}_8(\varphi) = \frac{1}{2}$ then $\langle \boldsymbol{\varepsilon}^x_1, 0; \lambda \rangle \models \exists z \varphi$ and $\langle \boldsymbol{\varepsilon}^x_1, 1; \lambda \rangle \models \exists z \varphi$ and $(T^{\exists z}_1, HC^{\exists z}_1) = (\{0; \lambda, 1; \lambda\}, \emptyset)$ is a $\Theta^{\varphi}_{\exists z}$-pair conditioned by $\boldsymbol{\varepsilon}^x_1$.*

If $Qz = \forall z$ instead, we get

- *Since $\boldsymbol{m}_3(\varphi) = 0$ and $\boldsymbol{m}_2(\varphi) = 0$ then $(T^{\forall z}_0, HC^{\forall z}_0) = (\emptyset, \emptyset)$ is a $\Theta^{\varphi}_{\forall z}$-pair conditioned by $\boldsymbol{\varepsilon}^x_0$;*
- *Since $\boldsymbol{m}_4(\varphi) = 1$ and $\boldsymbol{m}_4$ is a non-crisp interpration then $(T^{\forall z}_{\frac{1}{2}}, HC^{\forall z}_{\frac{1}{2}}) = (\emptyset, \{\boldsymbol{m}_4\})$ is a $\Theta^{\varphi}_{\forall z}$-pair conditioned by $\boldsymbol{\varepsilon}^x_{\frac{1}{2}}$;*
- *Since $\langle \boldsymbol{m}_7, \lambda \rangle \models \varphi$ and $\langle \boldsymbol{m}_9, \lambda \rangle \models \varphi$ and $\boldsymbol{m}_8(\varphi) = \frac{1}{2}$ then $\langle \boldsymbol{\varepsilon}^x_1, [0 \to \lambda \mid 1 \to \lambda] \rangle \models \forall z \varphi$ and $(T^{\forall z}_1, HC^{\forall z}_1) = (\{[0 \to \lambda \mid 1 \to \lambda]\}, \emptyset)$ is a $\Theta^{\varphi}_{\forall z}$-pair conditioned by $\boldsymbol{\varepsilon}^x_1$.*

Our algorithm, named *QEM*and reported in Algorithm 1, is a quantifier elimination algorithm from the outermost to the innermost quantifier. At each call, the *QEM* algorithm constructs as output a $\Theta^{\varphi}_{\overline{Q}}$-pair conditioned by $\boldsymbol{m}$. The output of the initial call $QEM(\overline{Q}, \varphi, \boldsymbol{\varepsilon})$ is a $\Theta^{\varphi}_{\overline{Q}}$-pair conditioned by $\boldsymbol{\varepsilon}$, (T, HC), such that T is the set of the equilibrium policies of $\overline{Q}\varphi$.

Base cases for the existential quantifier (lines 4 to 6) and the universal quantifier (lines 9 to 13) of the *QEM* algorithm apply Definition 8 to binders with only one quantifier following a restrictive application of minimality to the quantified variable:

- If $\langle \boldsymbol{m}, \pi \rangle$ is an equilibrium configuration of $\exists x \varphi$ then $\pi = v; \lambda$ with
  1. $v = 0$ and $\boldsymbol{m}^x_v(\varphi) = 1$ or

2. $v = 1$ and $\boldsymbol{m}^x_v(\varphi) = 1$ and $\boldsymbol{m}^x_{\frac{1}{2}}(\varphi) \leq 1$.

- If $\langle \boldsymbol{m}, \pi \rangle$ is an equilibrium configuration of $\forall x\varphi$ then $\pi = [0 \to \lambda \mid 1 \to \lambda]$ with
  1. $\boldsymbol{m}^x_0(\varphi) = 1$ and
  2. $\boldsymbol{m}^x_1(\varphi) = 1$ and $\boldsymbol{m}^x_{\frac{1}{2}}(\varphi) \leq 1$.

Equilibrium configurations cannot be constructed exclusively based on the interpretation $\boldsymbol{m}$. Clearly, the first item of Definition 8 imposes a more global condition: given a quantified formula $\overline{Q}\varphi$ and an equilibrium configuration $\langle \boldsymbol{m}, \pi \rangle$ of $\overline{Q}\varphi$, every $\boldsymbol{m}' \in_{\overline{Q}} \langle \boldsymbol{m}, \pi \rangle$ is an equilibrium model of φ. This condition is fulfilled in our algorithm thanks to the non-crisp part of the $\Theta^{\varphi}_{\overline{Q}}$-pair conditioned by the updates of the interpretation $\boldsymbol{m}$.

Example 5 (Example 4 continued)**.** *$\langle \boldsymbol{\varepsilon}^x_1, 0; \lambda \rangle \models \exists z\varphi$ but $\boldsymbol{m}_7 \in_{\exists z} \langle \boldsymbol{\varepsilon}^x_1, 0; \lambda \rangle$ and $\boldsymbol{m}_4 \lhd \boldsymbol{m}_7$, $\boldsymbol{m}_4$ is a non-crisp model of φ, element of the non-crisp part of $(\{0; \lambda\}, \{\boldsymbol{m}_4\})$, the $\Theta^{\varphi}_{\exists z}$-pair conditioned by $\boldsymbol{\varepsilon}^x_{\frac{1}{2}}$. So $\boldsymbol{m_7}$ is not an equilibrium model and $\langle \boldsymbol{\varepsilon}, 1; 0; \lambda \rangle \not\models \exists x \exists z\varphi$.*

We define two new operators, the $mc_{\exists}$ operator for $\exists$ quantifier and the $mc_{\forall}$ operator for $\forall$ quantifier, to combine, after the elimination of the quantifier Qx, the $\Theta^{\varphi}_{\overline{Q}}$-pairs (T_0, HC_0), $(T_{\frac{1}{2}}, HC_{\frac{1}{2}})$ and (T_1, HC_1) conditioned by resp. $\boldsymbol{m}^x_0$, $\boldsymbol{m}^x_{\frac{1}{2}}$ and $\boldsymbol{m}^x_1$, in a new $\Theta^{\varphi}_{Qx\overline{Q}}$-pairs conditioned by $\boldsymbol{m}$.

Definition 10 ($mc_{\exists}$ operator)**.** *Let $\overline{Q}$ be a binder and φ a quantifier-free formula and $x \in \mathtt{VAR}(\varphi) \setminus \mathtt{VAR}(\overline{Q})$. Let (T_0, HC_0), $(T_{\frac{1}{2}}, HC_{\frac{1}{2}})$, and (T_1, HC_1) some $\Theta_{\overline{Q}}$-pairs conditioned by the interpretations, respectively, $\boldsymbol{m}^x_0$, $\boldsymbol{m}^x_{\frac{1}{2}}$ and $\boldsymbol{m}^x_1$. The $mc_{\exists}$ operator is defined as follows: $mc_{\exists}(x, (T_0, HC_0), (T_{\frac{1}{2}}, HC_{\frac{1}{2}}), (T_1, HC_1)) = (T_{\exists x}, HC_{\exists x})$ where*

$$
\begin{aligned}
T_{\exists x} &= \{0; \pi \mid \pi \in T_0\} \\
&\cup \{1; \pi \mid \pi \in T_1,\ \boldsymbol{m}_1 \in_{\overline{Q}} \pi,\ \boldsymbol{m}_1 \triangleq_{\mathtt{VAR}(\varphi) \setminus \mathtt{VAR}(\exists x \overline{Q})} \boldsymbol{m}, \\
&\qquad \textit{there is no } \boldsymbol{m}' \in HC_{\frac{1}{2}} \textit{ s.t. } \boldsymbol{m}' \lhd \boldsymbol{m}_1 \textit{ and} \qquad (5) \\
&\qquad \textit{there is no } \pi' \in T_{\frac{1}{2}} \textit{ s.t. } \boldsymbol{m}_1 \in_{\overline{Q}} \pi'\} \qquad (6) \\
HC_{\exists x} &= \{\boldsymbol{m}' \mid \pi \in T_{\frac{1}{2}} \textit{ and } \boldsymbol{m}' \in_{\overline{Q}} \pi\} \cup \bigcup_{i \in \{0, \frac{1}{2}, 1\}} HC_i
\end{aligned}
$$

By Definition 9 and the definition of $\lhd$, if $\pi \in T_0$ then $0; \pi \in T_{\exists x}$. If there exists a $\boldsymbol{m}'$ such that (5) holds or π' is such that (6) holds, then $\boldsymbol{m}$ is no more $\trianglelefteq$-minimal and $\langle \boldsymbol{m}, 1; \pi \rangle$ cannot be an equilibrium configuration, otherwise $1; \pi \in T_{\exists x}$.

Example 6 (Example 4 continued)**.** *Given the QBF $\exists x \exists z\varphi$, the $mc_{\exists}$ applied on x returns*

$$
mc_{\exists}(x, (T^{\exists z}_0, HC^{\exists z}_0), (T^{\exists z}_{\frac{1}{2}}, HC^{\exists z}_{\frac{1}{2}}), (T^{\exists z}_1, HC^{\exists z}_1)) = (\{0; 0; \lambda, 1; 1; \lambda\}, \{\boldsymbol{m}_4\}).
$$

The $mc_{\forall}$ operator is similar to the $mc_{\exists}$ operator. The difference is that the $mc_{\forall}$ operator have simultaneously the both conditions on T_0 and T_1.

Definition 11 ($mc_{\forall}$ operator)**.** *Let $\overline{Q}$ be a binder and φ a quantifier-free formula and $x \in \mathtt{VAR}(\varphi) \setminus \mathtt{VAR}(\overline{Q})$. Let (T_0, HC_0), $(T_{\frac{1}{2}}, HC_{\frac{1}{2}})$, and (T_1, HC_1) some $\Theta_{\overline{Q}}$-pairs conditioned by the interpretations*

respectively, $\boldsymbol{m}_0^x$, $\boldsymbol{m}_{\frac{1}{2}}^x$ *and* $\boldsymbol{m}_1^x$. *The* $mc_\forall$ *operator is defined as follows:* $mc_\forall(x,(T_0,HC_0),(T_{\frac{1}{2}},HC_{\frac{1}{2}}),(T_1,HC_1)) = (T_{\forall x},HC_{\forall x})$ *where*

$$\begin{aligned} T_{\forall x} \;=\; \{[0\to\pi_0 \mid 1\to\pi_1] \mid\; & \pi_0\in T_0,\ \pi_1\in T_1,\\ & \boldsymbol{m}_1 \in_{\overline{Q}} \pi_1,\ \boldsymbol{m}_1 \triangleq_{\mathtt{VAR}(\varphi)\setminus\mathtt{VAR}(\forall x\overline{Q})} \boldsymbol{m},\\ & \textit{there is no } \boldsymbol{m}'\in HC_{\frac{1}{2}} \textit{ s.t. } \boldsymbol{m}'\lhd\boldsymbol{m}_1 \textit{ and}\\ & \textit{there is no } \pi'\in T_{\frac{1}{2}} \textit{ s.t. } \boldsymbol{m}_1\in_{\overline{Q}}\pi'\} \end{aligned}$$

and $HC_{\forall x}$ *is defined as* $HC_{\exists x}$.

Example 7 (Example 4 continued)**.** *Given* $\varphi = ((z\to x)\wedge(z\vee\neg z))$*, we conclude that, for the formula* $(\forall x\exists z\varphi)$*,*

$$mc_\forall(x,(T_0^{\exists z},HC_0^{\exists z}),(T_{\frac{1}{2}}^{\exists z},HC_{\frac{1}{2}}^{\exists z}),(T_1^{\exists z},HC_1^{\exists z})) = (\{[0\to 0;\lambda \mid 1\to 1;\lambda]\},\{\boldsymbol{m}_4\}).$$

Algorithm 1: *QEM*

input : A non empty binder $\overline{Q}$, a propositional formula φ and a G_3 interpretation $\boldsymbol{m}$.
output: A $\Theta_{\overline{Q}}^{\varphi}$-pair conditioned by $\boldsymbol{m}$

$T_x \leftarrow \emptyset$;
$HC_x \leftarrow \emptyset$;
if $\overline{Q} = \exists x$ **then**
  if $\boldsymbol{m}_{\frac{1}{2}}^x(\varphi) = 1$ **then** $HC_x \leftarrow HC_x \cup \{\boldsymbol{m}_{\frac{1}{2}}^x\}$;
  if $\boldsymbol{m}_0^x(\varphi) = 1$ **then** $T_x \leftarrow T_x \cup \{0;\lambda\}$;
  if $\boldsymbol{m}_{\frac{1}{2}}^x(\varphi) \le \frac{1}{2}$ *and* $\boldsymbol{m}_1^x(\varphi) = 1$ **then** $T_x \leftarrow T_x \cup \{1;\lambda\}$;
  return (T_x,HC_x);
if $\overline{Q} = \forall x$ **then**
  if $\boldsymbol{m}_0^x(\varphi) = 1$ *and* $\boldsymbol{m}_{\frac{1}{2}}^x(\varphi) \le \frac{1}{2}$ *and* $\boldsymbol{m}_1^x(\varphi) = 1$ **then return** $(\{[0\to\lambda \mid 1\to\lambda]\},\emptyset)$;
  if $\boldsymbol{m}_{\frac{1}{2}}^x(\varphi) = 1$ **then** $HC_x \leftarrow HC_x \cup \{\boldsymbol{m}_{\frac{1}{2}}^x\}$;
  if $\boldsymbol{m}_0^x(\varphi) = 1$ *and* $\boldsymbol{m}_0^x$ *is not crisp* **then** $HC_x \leftarrow HC_x \cup \{\boldsymbol{m}_0^x\}$;
  if $\boldsymbol{m}_1^x(\varphi) = 1$ *and* $\boldsymbol{m}_1^x$ *is not crisp* **then** $HC_x \leftarrow HC_x \cup \{\boldsymbol{m}_1^x\}$;
  return $(\emptyset,HC_x)$;
if $\overline{Q} = Qx\overline{Q'}$ **then**
  $(T_0,HC_0) = QEM(\overline{Q'},\varphi,\boldsymbol{m}_0^x)$;
  $(T_{\frac{1}{2}},HC_{\frac{1}{2}}) = QEM(\overline{Q'},\varphi,\boldsymbol{m}_{\frac{1}{2}}^x)$;
  $(T_1,HC_1) = QEM(\overline{Q'},\varphi,\boldsymbol{m}_1^x)$;
  if $Q = \exists$ **then return** $mc_\exists(x,(T_0,HC_0),(T_{\frac{1}{2}},HC_{\frac{1}{2}}),(T_1,HC_1))$;
  else return $mc_\forall(x,(T_0,HC_0),(T_{\frac{1}{2}},HC_{\frac{1}{2}}),(T_1,HC_1))$;

Theorem 12. *Given a QBF* $\overline{Q}\varphi$ *and a* G_3 *interpretation* $\boldsymbol{m}$, $QEM(\overline{Q},\varphi,\boldsymbol{m})$ *computes the set of equilibrium policies of* $\overline{Q}\varphi$ *conditioned by* $\boldsymbol{m}$.

Example 8 (Examples 6 and 7 continued)**.** *With* $\varphi = ((z\to x)\wedge(z\vee\neg z))$ *we have that*

1. *$\exists x \exists z \varphi$ has two equilibrium policies:* $0;0;\lambda$ *and* $1;1;\lambda$*. The reader can verify that the formula $\exists z \exists x \varphi$ has the same equilibrium policies. This result is expected since, as in QBFs two quantifiers in the same block (therefore, with the same type) can be reversed without affecting the satisfiability.*
2. *$\forall x \exists z \varphi$ has the equilibrium policy* $[0 \to 0;\lambda \mid 1 \to 1;\lambda]$*. In this case, the formula $\forall z \exists x \varphi$ has the same equilibrium policies. However, as in QBFs, this is not generally true.*
3. *Finally, the reader can verify that the formulas $\exists x \forall z \varphi$, $\exists z \forall x \varphi$, $\forall z \forall x \varphi$ and $\forall x \forall z \varphi$ have no equilibrium policies.*

Algorithm 1 computes all equilibrium policies associated to an equilibrium model. We provide some comments on our algorithm. The base cases, $\exists x\ \varphi$ and $\forall x\ \varphi$ consider only total G_3 interpretations that satisfy φ and cannot be discarded by (only) minimising the truth of x. Also, non-total interpretations that satisfy φ will provide no equilibrium policy at all but they will be kept since they can be used to disregard other interpretations. For the inductive step, the operator $mc_\exists$ (resp. $mc_\forall$) applies the minimisation condition (Definition 10.($T_{\exists x}$) (resp. Definition 11.($T_{\forall x}$))) in order to discard equilibrium policies that are not minimal with respect to Definition 3. Also potential interpretations that can be used to discard total models are kept for further tests. For sake of regularity, in $T_{\frac{1}{2}}$, QBF policies are built from non-crisp models but, of course, all those models can be put into the set of non-crisp models $HC_{\frac{1}{2}}$ and $T_{\frac{1}{2}}$ left empty. For sake of simplicity, the set of non-crisp models is not managed optimally but the relation $\trianglelefteq$ may be used to keep only the minimal elements. A `Prolog` prototype is available at `https://leria-info.univ-angers.fr/~igor.stephan/Research/EQUILIBRIUM_POLICIES/equilibrium_policies.html`.

5 Conclusions and Future Work

In this paper, we considered Quantified Answer Set Programming (QASP) as defined in [20, 11], where propositional quantification is applied over propositional variables. We demonstrated that the two referenced semantics are not equivalent. We also addressed the problem of policy generation [8] under the semantics of [11], introducing an algorithm to compute all minimal policies associated with a QBF. Our approach relies on quantified propositional G_3 [3] as a monotonic base and uses ideas from equilibrium logic for minimality. The algorithm has been implemented in a `Prolog` prototype, allowing practical evaluation on real examples. For future work, we plan to explore the following directions:

1. **Stéphan's semantics:** The interpretation of propositional quantification as filters over stable models [11] aligns with ASP, but restricts players from explicitly choosing truth assignments. Stéphan's approach allows such moves; we aim to further explore its potential in Knowledge Representation.
2. **Algorithmic improvements:** Our current algorithm exhaustively explores all cases, including potentially redundant ones. We intend to first explore the complexity of the algorithm, which we did not consider in this paper, as well as to study the formal properties of our framework to simplify and optimize the procedure.
3. **Syntactic subfragments of QBF:** Given that QASP satisfiability is PSPACE-complete [11], we are interested in identifying syntactic restrictions that may yield lower complexity, following insights from [21, 17].
4. **Application to two-player games:** QASP has been applied to model two-player games [13]. We plan to apply our policy-extraction algorithm in this context, moving beyond satisfiability checking to identify concrete strategies.

References

[1] G. Amendola, B. Cuteri, F. Ricca & M. Truszczynski (2022): *Solving Problems in the Polynomial Hierarchy with ASP(Q)*. In G. Gottlob, D. Inclezan & M. Maratea, editors: *Proceedings of the Sixteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'22)*, *Lecture Notes in Artificial Intelligence* 13416, Springer-Verlag, pp. 373–386, doi:10.1007/978-3-031-15707-3_29.

[2] G. Amendola, F. Ricca & M. Truszczynski (2019): *Beyond NP: Quantifying over Answer Sets*. *Theory and Practice of Logic Programming* 19(5-6), p. 705–721, doi:10.1017/S1471068419000140.

[3] M. Baaz, A. Ciabattoni & R. Zach (2000): *Quantified Propositional Gödel Logics*. In: *Proceedings of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR)*, *LNCS* 1955, Springer, pp. 240–256, doi:10.1007/3-540-44404-1_16.

[4] M. Benedetti (2005): *sKizzo: a Suite to Evaluate and Certify QBFs*. In: *Proceedings of the 20th International Conference on Automated Deduction (CADE'05)*, pp. 369–376, doi:10.1007/11532231_27.

[5] A. Biere, M. Heule, H. van Maaren & T. Walsh, editors (2021): *Handbook of Satisfiability*. *Frontiers in Artificial Intelligence and Applications* 336, IOS Press, doi:10.3233/FAIA336.

[6] B. Bogaerts, T. Janhunen & S. Tasharrofi (2016): *Stable-Unstable Semantics: Beyond NP With Normal Logic Programs*. *Theory and Practice of Logic Programming* 16(5-6), pp. 570–586, doi:10.1017/S1471068416000387.

[7] G. Brewka, T. Eiter & M. Truszczyński (2011): *Answer set programming at a glance*. *Communications of the ACM* 54(12), pp. 92–103, doi:10.1145/2043174.2043195.

[8] S. Coste-Marquis, H. Fargier, J. Lang, D. Le Berre & P. Marquis (2006): *Representing Policies for Quantified Boolean Formulae*. In P. Doherty, J. Mylopoulos & C. Welty, editors: *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, AAAI Press, pp. 286–297.

[9] W. Faber (2024): *Solving Argumentation Problems Using Answer Set Programming with Quantifiers: Preliminary Report*. In: *Workshop Proceedings of the 40th International Conference on Logic Programming (ICLP-WS)*, *CEUR Workshop Proceedings* 3799.

[10] W. Faber & M. Morak (2023): *Evaluating Epistemic Logic Programs via Answer Set Programming with Quantifiers*. In: *Thirty-Seventh AAAI Conference on Artificial Intelligence*, pp. 6322–6329, doi:10.1609/AAAI.V37I5.25778.

[11] J. Fandinno, F. Laferriere, J. Romero, T. Schaub & T. Son (2021): *Planning with Incomplete Information in Quantified Answer Set Programming*. *Theory and Practice of Logic Programming* 21(5), pp. 663–679, doi:10.1017/S1471068421000259.

[12] K. Gödel (1932): *Zum intuitionistischen Aussagenkalkül*. *Anzeiger der Akademie der Wissenschaften in Wien*, pp. 65–66.

[13] Y. He, A. Saffidine & M. Thielscher (2024): *Solving Two-player Games with QBF Solvers in General Game Playing*. In: *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, IFAAMAS, p. 807–815, doi:10.5555/3635637.3662934.

[14] A. Heyting (1930): *Die formalen Regeln der intuitionistischen Logik*. In: *Sitzungsberichte der Preussischen Akademie der Wissenschaften*, Deutsche Akademie der Wissenschaften zu Berlin, pp. 42–56.

[15] V. Lifschitz (2010): *Thirteen Definitions of a Stable Model*. In: *Fields of Logic and Computation: Essays Dedicated to Yuri Gurevich on the Occasion of His 70th Birthday*, Springer, pp. 488–503, doi:10.1007/978-3-642-15025-8_24.

[16] G. Malinowski (1993): *Many-valued logics*. *Oxford logic guides* , Oxford University Press,Clarendon Press, doi:10.1093/oso/9780198537878.001.0001.

[17] N. Oijid (2024): *Bounded degree QBF and positional games*. *CoRR*, doi:10.48550/ARXIV.2411.10093.

[18] D. Pearce (2006): *Equilibrium logic*. *Annals of Mathematics and Artificial Intelligence* 47(1-2), pp. 3–41, doi:10.1007/S10472-006-9028-Z.

[19] D. Pearce, H. Tompits & S. Woltran (2001): *Encodings for Equilibrium Logic and Logic Programs with Nested Expressions*. In: *Proceedings of the Tenth Portuguese Conference on Artificial Intelligence (EPIA'01)*, *Lecture Notes in Computer Science* 2258, Springer-Verlag, pp. 306–320, doi:10.1007/3-540-45329-6_31.

[20] I. Stéphan (2014): *QASP ou la programmation par ensembles réponses quantifiée*. In: *Actes du dix-neuvième congrès national sur la Reconnaissance de Formes et l'Intelligence Artificielle (RFIA)*. Available at https://hal.science/hal-00989206v1/document. In French.

[21] I. Stéphan & B. Da Mota (2009): *A Unified Framework for Certificate and Compilation for QBF*. In: *Proceedings of the Third Indian Conference on Logic and its Applications (ICLA)*, *LNCS* 5378, Springer, pp. 210–223, doi:10.1007/978-3-540-92701-3_15.

[22] L.J. Stockmeyer & A.R. Meyer (1973): *Word problems requiring exponential time*. In: *Proceedings of the 5th annual ACM symposium on Theory of computing (STOC)*, pp. 1–9, doi:10.1145/800125.804029.

A Simple and Effective ASP-Based Tool for Enumerating Minimal Hitting Sets

Mohimenul Kabir
National University of Singapore
School of Computing
Singapore

Kuldeep S Meel
Georgia Institute of Technology
School of Computer Science
USA

The hitting set problem is a fundamental problem in computer science and mathematics. Given a family of sets over a universe of elements, a minimal hitting set is a subset-minimal collection of elements that intersects each set in the family. Enumerating all minimal hitting sets is crucial in various real-world applications.

In this paper, we address the full enumeration of all minimal hitting sets for a given family of sets. We formulate the problem using Answer Set Programming (ASP) and leverage existing ASP solvers for efficient enumeration. We propose an ASP-based tool, MinHit-ASP, and our empirical evaluation shows that it effectively enumerates minimal hitting sets across benchmarks from diverse problem domains.

Keywords. Tool: MinHit-ASP, Answer Set Solving, Minimal Hitting Set, Answer Set Enumeration.

1 Introduction

Given a family of sets $\mathcal{S} = \{S_1, \ldots, S_k\}$, a *hitting set* of $\mathcal{S}$ is a set h that intersects every set of $\mathcal{S}$ [65]. A hitting set h is called *minimal hitting set* (MHS) if no proper subset of h is a hitting set of $\mathcal{S}$. In the MHS problem, each $S_i \in \mathcal{S}$ is a *set* and the union $\mathcal{U} = \bigcup_i S_i$ is called the *universe*. The problem definition is closely related to *minimal hypergraph traversal* [29]. The minimal hitting sets have applications in a wide range of domains, such as diagnosis [1, 5, 59], explanability [33, 60], data mining [4, 13, 29], bioinformatics and computational biology [44, 62], and combinatorial games [55].

In many cases, a family of sets $\mathcal{S}$ has multiple minimal hitting sets (MHSes); in fact, the number of MHSes can be exponential [15]. In such settings, enumerating all MHSes is often desirable to gain a deeper understanding of the underlying problem. For instance, consider a faulty system where each *diagnosis* corresponds to a hitting set of the problem [59]. Enumerating all hitting sets provides a complete characterization of the possible fault diagnoses. Similarly, the OCSANA framework [64] computes all MHSes of *elementary paths* to analyze *signal transduction networks*.

A wide range of techniques have been developed for computing minimal hitting sets (MHSes). To the best of our knowledge, the first attempt to compute MHSes was introduced by Berge [7] in the context of hypergraph traversal. Later, Reiter proposed a method for computing MHSes using a specialized data structure known as the *hitting set tree* [59]. Over the years, numerous MHS algorithms have been developed across various domains, including hypergraph traversal [7, 57], model-based diagnosis [28], data mining [4, 17], poset theory [48], Boolean algebra [22], integer programming [12], binary decision diagrams [61], and evolutionary computation [49]. In contrast, we adopt an alternative approach by formulating MHS enumeration as an *answer set program* [25].

Answer Set Programming (ASP) [52] is a well-established declarative paradigm in knowledge representation and reasoning, widely used for modeling complex combinatorial problems [21]. An ASP

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 238–247, doi:10.4204/EPTCS.439.17

© Kabir and Meel
This work is licensed under the
Creative Commons Attribution License.

program consists of *logical rules (grounded)* over *propositional atoms*, encoding domain knowledge and queries. A solution to an ASP program, known as an *answer set*, is an assignment of values to propositional atoms that satisfies the program's semantics. ASP programs are classified into two categories based on their expressiveness: *disjunctive* and *normal* ASP programs [51]. However, the higher expressiveness comes at the cost of computational complexity. Determining whether a disjunctive ASP program has an answer set is $\sum_2^P$-complete [20], whereas for normal ASP programs, the complexity reduces to NP-complete [53].

The main contribution of this paper is the design, implementation, and experimental evaluation of a new MHS enumeration tool called MinHit-ASP. MinHit-ASP exploits the efficiency of ASP solvers for the MHS enumeration. More specifically, MinHit-ASP reduces MHS enumeration to answer set enumeration problem of a *head-cycle free* disjunctive answer set program [63]. Note that the MHS problem has also been reduced to ASP solving in earlier work [2]. According to the computational complexity, the minimal hitting set problem is NP-hard [42, 58]. The decision problem for disjunctive logic programs lies in $\sum_2^P$ [20]. It is worth noting that reducing one problem to a harder one is not uncommon in combinatorial problem-solving [46]. It is well known to exploit ASP solvers in real-world diagnosis problems [30, 66] and related problems [39, 40]. To the end, we conduct experimental evaluation to evaluate MinHit-ASP on real-world hitting set benchmarks. Our extensive evaluation reveals that MinHit-ASP outperforms existing systems of all MHSes enumeration problem.

The remainder of this paper is organized as follows: Section 2 provides the necessary background to understand our main contribution. Section 3 describes the methodology and implementation of MinHit-ASP. Section 4 presents the experimental evaluation of MinHit-ASP on real-world benchmark datasets. Finally, we conclude our study in Section 5.

2 Preliminaries

Answer Set Programming. An *answer set program* P consists of a set of rules, each rule is structured as follows:

$$\text{Rule } r\text{: } a_1 \vee \ldots \vee a_k \leftarrow b_1, \ldots, b_m, \mathsf{not}\ c_1, \ldots, \mathsf{not}\ c_n \tag{1}$$

where, $a_1, \ldots, a_k, b_1, \ldots, b_m, c_1, \ldots, c_n$ are *propositional variables* or *atoms*, and k, m, n are non-negative integers. The notations P and $\mathsf{atoms}(P)$ denote the rules and atoms within the program P. In rule r, the operator "not" denotes *default negation* [16]. For each rule r (eq. (1)), we adopt the following notations: the atom set $\{a_1, \ldots, a_k\}$ constitutes the *head* of r, denoted by $\mathsf{Head}(r)$, the set $\{b_1, \ldots, b_m\}$ is referred to as the *positive body atoms* of r, denoted by $\mathsf{Body}(r)^+$, and the set $\{c_1, \ldots, c_n\}$ is referred to as the *negative body atoms* of r, denoted by $\mathsf{Body}(r)^-$. We often represent empty body ($\mathsf{Body}(r)^+ = \emptyset$ and $\mathsf{Body}(r)^- = \emptyset$) by the notation $\top$. A program P is called a *disjunctive logic program* if $\exists r \in P$ such that $|\mathsf{Head}(r)| \geq 2$ [6]. In this paper, we use the ASP-Core-2 standard input language for ASP programs [14].

In ASP, an interpretation M over $\mathsf{atoms}(P)$ specifies which atoms are assigned true; that is, an atom a is true under M if and only if $a \in M$ (or false when $a \notin M$ resp.). An interpretation M satisfies a rule r, denoted by $M \models r$, if and only if $(\mathsf{Head}(r) \cup \mathsf{Body}(r)^-) \cap M \neq \emptyset$ or $\mathsf{Body}(r)^+ \setminus M \neq \emptyset$. An interpretation M is a *model* of P, denoted by $M \models P$, when $\forall_{r \in P} M \models r$. The *Gelfond-Lifschitz (GL) reduct* of a program P, with respect to an interpretation M, is defined as $P^M = \{\mathsf{Head}(r) \leftarrow \mathsf{Body}(r)^+ | r \in P, \mathsf{Body}(r)^- \cap M = \emptyset\}$ [27]. An interpretation M is an *answer set* of P if $M \models P$ and no $M' \subsetneq M$ exists such that $M' \models P^M$. We denote the answer sets of program P using the notation $\mathsf{AS}(P)$.

1 $g(a_1) \vee \ldots \vee g(a_\ell) \leftarrow \top.$

Encoding 1: The operator $\mathsf{HS2ASP}(\mathcal{S})$ for a family of sets $\mathcal{S}$. The lines starting with % sign are comments. We often use the notation $\top$ to denote empty rule body.

3 Methodology and Architecture

In this section, we present the methodology and architecture of MinHit-ASP. At a high level, MinHit-ASP transforms the MHS enumeration problem into an ASP solving task. We first describe the reduction process from MHS enumeration to ASP solving. Then, we outline the architecture of MinHit-ASP.

From MHS enumeration to ASP Solving. Given a family of sets $\mathcal{S}$, we define an operator $\mathsf{HS2ASP}(\mathcal{S})$ that constructs an ASP program. In the reduction to ASP solving, given the universe $\mathcal{U} = \bigcup_i S_i$, we introduce a mapping $g(\cdot)$, which maps each element $x \in \mathcal{U}$ to a unique propositional atom $g(x)$. The operator $\mathsf{HS2ASP}(\mathcal{S})$ is applied to each set $S_i = \{a_1, \ldots, a_\ell\} \in \mathcal{S}$, generating a corresponding rule $g(a_1) \vee \ldots \vee g(a_\ell) \leftarrow \top.$, as illustrated in Encoding 1. The intuition behind introducing $\mathsf{HS2ASP}(\mathcal{S})$ is that each model of $\mathsf{HS2ASP}(\mathcal{S})$ corresponds to a hitting set of $\mathcal{S}$, and vice versa. Moreover, each answer set of $\mathsf{HS2ASP}(\mathcal{S})$ corresponds precisely to a minimal hitting set of $\mathcal{S}$. We formally prove that the answer sets of $\mathsf{HS2ASP}(\mathcal{S})$ correspond one-to-one with the minimal hitting sets of $\mathcal{S}$.

Lemma 1. *For a family of sets $\mathcal{S}$,*

1. *each minimal hitting set of $\mathcal{S}$ corresponds to an answer set of $\mathsf{HS2ASP}(\mathcal{S})$.*
2. *each answer set of $\mathsf{HS2ASP}(\mathcal{S})$ corresponds to a minimal hitting set of $\mathcal{S}$.*

Proof. **Part of '1'**: Without loss of generality, assume that $h = \{a_1, \ldots, a_\ell\}$ is a minimal hitting set of $\mathcal{S}$. We show that $M = \{g(a_1), \ldots, g(a_\ell)\}$ is an answer set of $\mathsf{HS2ASP}(\mathcal{S})$.

Note that $M \models \mathsf{HS2ASP}(\mathcal{S})$ and $\mathsf{HS2ASP}(\mathcal{S})^M = \mathsf{HS2ASP}(\mathcal{S})$, since there is no negation (not) in $\mathsf{HS2ASP}(\mathcal{S})$. Clearly, there is no $M' \subsetneq M$ such that $M' \models \mathsf{HS2ASP}(\mathcal{S})^M$. Otherwise, h is not a minimal hitting set. Thus, M is an answer set of $\mathsf{HS2ASP}(\mathcal{S})$.

Part of '2': Assume that there is an answer set $M = \{g(a_1), \ldots, g(a_\ell)\} \in \mathsf{AS}(\mathsf{HS2ASP}(\mathcal{S}))$. We show that $h = \{a_1, \ldots, a_\ell\}$ is a minimal hitting set of $\mathcal{S}$.

According to the answer set semantics, $\forall S_i \in \mathcal{S}$, $S_i \cap h \neq \emptyset$, i.e., h is a hitting set of $\mathcal{S}$. Clearly, there is no $h' \subsetneq h$; otherwise, from the hitting set h', $\exists M' \subset M$ such that $M' \models \mathsf{HS2ASP}(\mathcal{S})^M$, thus $M \notin \mathsf{AS}(\mathsf{HS2ASP}(\mathcal{S}))$ — a contradiction. So, h is a minimal hitting set of $\mathcal{S}$. □

Architecture of MinHit-ASP. The architecture of MinHit-ASP is illustrated in Figure 1. Given a family of sets $\mathcal{S}$, MinHit-ASP constructs the corresponding ASP program $\mathsf{HS2ASP}(\mathcal{S})$, as defined in Encoding 1. It then invokes an ASP solver to enumerate the answer sets of $\mathsf{HS2ASP}(\mathcal{S})$.

Notably, although the disjunctive ASP program $\mathsf{HS2ASP}(\mathcal{S})$ can be translated into a normal logic program [47], this translation results in a quadratic blowup [2].

Example 1. *Consider the family of sets $\mathcal{S} = \{\{1,2\},\{3\},\{2,3,4\}\}$. The sets $\mathcal{S}$ has two minimal hitting sets: $\{1,3\}$ and $\{2,3\}$.*

The ASP program $\mathsf{HS2ASP}(\mathcal{S})$ consists of rules: $\{g(1) \vee g(2) \leftarrow \top.\ g(3) \leftarrow \top.\ g(2) \vee g(3) \vee g(4) \leftarrow \top.\}$, and it has two answer sets: $\{g(1), g(3)\}$ and $\{g(2), g(3)\}$, which correspond to two MHSes of $\mathcal{S}$.

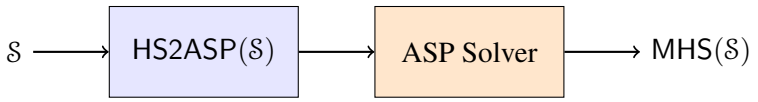


Figure 1: The high-level methodology of MinHit-ASP.

4 Experimental Evaluation

We present the experimental evaluation of MinHit-ASP on publicly available benchmarks from various domains. The primary objective is to assess how well MinHit-ASP performs compared to existing tools for minimal hitting set enumeration. The dataset and code used in our experimental evaluation are available at: `https://zenodo.org/records/15108391`.

Design Choice We employed Clingo v5.7.1 [24, 26] as the underlying ASP solver in MinHit-ASP. We also evaluated Wasp v2 [3] (with the parameter `-disjunction=auto`), but we observed that Wasp did not outperform Clingo in our experiments[1].

Benchmark and Baseline Our benchmark suite consists of instances drawn from real-world domains involving minimal hitting set problems, including unique column combinations [9], cluster vertex deletion [8], metabolic reactions [31], cell signaling networks [67], the Connect-4 board game [18], frequent itemset mining [57], graph theory [11, 43], and combinatorial circuits from the ISCAS85 suite [45]. We also include randomly generated instances from [57]. The benchmark set is primarily compiled from datasets used in [10, 57]. These benchmarks contain up to 3,511 elements and 1,973,734 sets.

We compare MinHit-ASP against several existing systems capable of enumerating minimal hitting sets. Specifically, we evaluate the performance of Hitman [34], Sparsity-based Hypergraph Dualization (SHD) [57], and MtMiner [32]. For Hitman, we use the MCS-based technique [34, 56] (this setting enumerates MHSes in an unordered way, which makes for the fairest comparison.). We also include SAT-based [19, 50] and Integer Linear Programming (ILP)-based approaches in our comparison. These techniques iteratively invoke a SAT solver and the Gurobi ILP solver (v11.0.1), respectively, to enumerate all minimal hitting sets. Since MHSes correspond to *minimal models*, we do not compare against tools designed specifically for full AllSAT enumeration, such as HALL [23] and the method from [54].

Environmental Settings All experiments were conducted on a high-performance computing cluster, with each node consisting of Intel Xeon Gold 6248 CPUs. Each benchmark instance was executed on a single core, with a time limit of 1000 seconds and a memory limit of 16 GB for all considered tools.

	SAT	ILP	MtMiner	SHD	Hitman	MinHit-ASP
#Solved	1735	647	1002	633	1846	**1948**

Table 1: The performance comparison of MinHit-ASP vis-a-vis existing hitting set solver.

Experimental Results We compare the number of benchmark instances solved by various MHS solvers, with the results summarized in Table 1. An instance is considered solved by a tool if it enumerates all

[1] We observed that Clingo and Wasp are able to solve 1948 and 1806 instances, respectively.

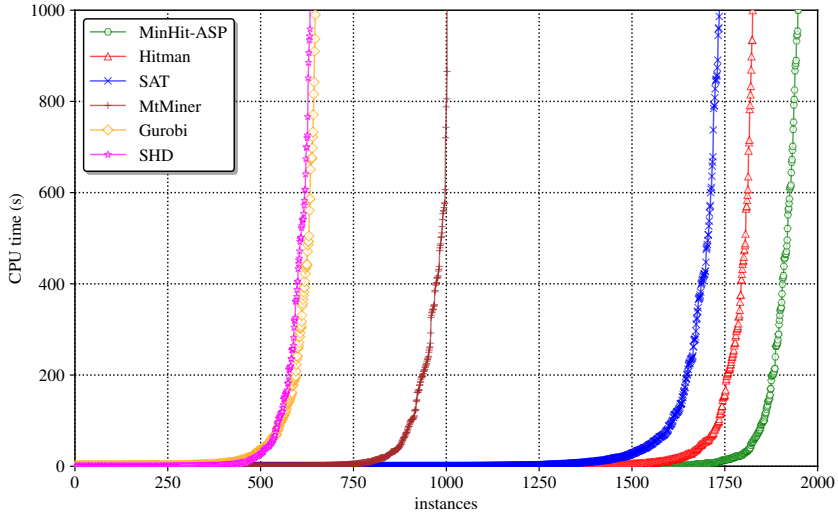
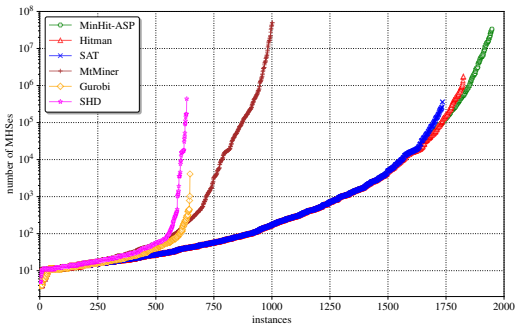
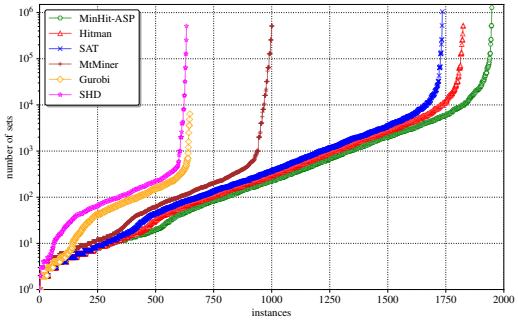


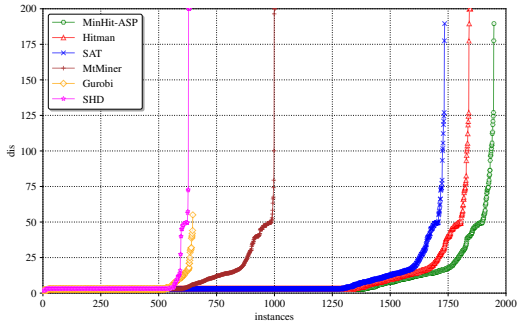
Figure 2: The runtime comparison of different minimal hitting set enumerators.



(a) The performance comparison in terms of the number of minimal hitting sets enumerated.



(b) The performance comparison in terms of the size of instances.



(c) The performance comparison in terms of the average disjunction size.

Figure 3: The performance comparison of MinHit-ASP vis-a-vis existing minimal hitting set solvers.

	VBS1	VBS2
#Solved	1906	2000

Table 2: The performance comparison of virtual best solvers. The virtual best solver VBS1 combines all baselines except MinHit-ASP and VBS2 combines all baselines including MinHit-ASP.

	$\sum$	SAT	ILP	MtMiner	SHD	Hitman	MinHit-ASP
$1 \leq \lvert\mathcal{S}\rvert < 10^1$	3	3	3	0	1	3	3
$10^1 \leq \lvert\mathcal{S}\rvert < 10^2$	320	285	141	225	61	302	**315**
$10^2 \leq \lvert\mathcal{S}\rvert < 10^3$	1312	397	280	376	280	426	**483**
$10^3 \leq \lvert\mathcal{S}\rvert < 10^4$	595	529	216	341	264	532	**533**
$10^4 \leq \lvert\mathcal{S}\rvert < 10^5$	524	452	7	25	16	478	**494**
$10^5 \leq \lvert\mathcal{S}\rvert < 10^6$	187	60	0	21	7	93	**110**
$10^6 \leq \lvert\mathcal{S}\rvert < 10^7$	69	8	0	14	4	9	9
$10^7 \leq \lvert\mathcal{S}\rvert < 10^8$	4	1	0	0	0	**3**	1

Table 3: The performance comparison of MinHit-ASP vis-a-vis existing hitting set solvers across different ranges of $|\mathcal{S}|$. The column labeled $\sum$ indicates the number of instances within each range.

minimal hitting sets within the specified time and memory limits. As shown in the table, MinHit-ASP outperforms existing hitting set solvers by a significant margin. The runtime performance comparison between MinHit-ASP and other MHS solvers is illustrated in Figure 2. A point (x,y) on these plots indicates that the tool solved x instances, each within y seconds.

To demonstrate the effectiveness of MinHit-ASP, we constructed virtual best solvers (VBS) combining MinHit-ASP and other baseline methods. Specifically, we define two VBS configurations: (i) VBS1, which includes all baselines excluding MinHit-ASP and (ii) VBS2, which includes all baselines including MinHit-ASP. The performance comparison is presented in Table 2. Notably, VBS2 solved 94 instances that were not solved by any of the other baselines alone, highlighting the unique contributions of MinHit-ASP. A deeper analysis shows that SAT, ILP, MtMiner, SHD, and Hitman respectively solved $7, 0, 41, 9$, and 42 instances not solved by MinHit-ASP. In contrast, MinHit-ASP solved $220, 1301, 987, 1234$, and 123 instances that were not solved by SAT, ILP, MtMiner, SHD, and Hitman, respectively.

Additionally, we compare the performance of minimal hitting set solvers based on three factors: the number of minimal hitting sets enumerated, the size of the instances, and the average size of the *disjunction*. These comparisons are visualized in Figures 3a to 3c. The instance size is defined as the number of sets in $\mathcal{S}$, denoted by $|\mathcal{S}|$. The average disjunction size, denoted by dis, is defined as $\frac{\sum_{S_i \in \mathcal{S}} |S_i|}{|\mathcal{S}|}$, representing the mean number of elements in each set S_i. A point (x,y) on these plots indicates that a solver successfully handled x instances, where each instance has at most y minimal hitting sets (in Figure 3a), has a size ($|\mathcal{S}|$) of at most y (in Figure 3b) or features an average disjunction size of at most y (in Figure 3c). The plots show that MinHit-ASP is competitive with other minimal hitting set solvers in terms of the number of MHSes enumerated, the size of the instances, and the average disjunction size it can handle.

	$\sum$	SAT	ILP	MtMiner	SHD	Hitman	MinHit-ASP
$10^1 \leq \mathsf{dis} < 10^2$	1788	1447	590	718	582	1505	**1563**
$10^2 \leq \mathsf{dis} < 10^3$	1145	278	57	279	46	325	**372**
$10^3 \leq \mathsf{dis} < 10^4$	76	10	0	2	1	**14**	13
$10^4 \leq \mathsf{dis} < 10^5$	5	0	0	3	**4**	2	0

Table 4: The performance comparison of MinHit-ASP vis-a-vis existing hitting set solvers across different ranges of the average disjunction size. The notation dis denotes the average size of disjunctions. The column labeled $\sum$ indicates the number of instances within each range.

We compare the performance of MinHit-ASP against other MHS solvers across varying ranges of instance size ($|\mathcal{S}|$) and average disjunction size (dis). The results are shown in Tables 3 and 4, where dis denotes the average disjunction size. Both tables show that MinHit-ASP struggles with instances that have larger $|\mathcal{S}|$ or higher dis; the number of solved instances decreases as $|\mathcal{S}|$ or dis increases. This performance drop may stem from the fact that MinHit-ASP reduces MHS enumeration to a harder problem than the other solvers do.

5 Conclusion

In this paper, we introduce MinHit-ASP, an ASP-based tool for minimal hitting set enumeration. Our ASP encoding is simple yet expressive, supporting versatile features such as enumerating minimal hitting sets of a *specified size, containing a given set of items, and optimal hitting sets with respect to objective functions*. Our tool also supports advanced reasoning tasks over hitting sets, including *brave*, *cautious* reasoning, and counting [35, 36, 37, 38, 41]. By leveraging a reduction to ASP, MinHit-ASP takes advantage of advancements in ASP solving. Our experimental evaluation demonstrates that MinHit-ASP outperforms existing minimal hitting set solvers in enumerating all MHSes.

References

[1] Rui Abreu & Arjan JC Van Gemund (2009): *A Low-Cost Approximate Minimal Hitting Set Algorithm and its Application to Model-Based Diagnosis*. In: *SARA*, 9, pp. 2–9.

[2] Mario Alviano & Carmine Dodaro (2016): *Completion of Disjunctive Logic Programs*. In: *IJCAI*, 16, pp. 886–892.

[3] Mario Alviano, Carmine Dodaro, Nicola Leone & Francesco Ricca (2015): *Advances in WASP*. In: *LPNMR*, Springer, pp. 40–54, doi:10.1007/978-3-319-23264-5_5.

[4] James Bailey, Thomas Manoukian & Kotagiri Ramamohanarao (2003): *A fast algorithm for computing hypergraph transversals and its application in mining emerging patterns*. In: *ICDM*, IEEE, pp. 485–488, doi:10.1109/ICDM.2003.1250958.

[5] James Bailey & Peter J Stuckey (2005): *Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization*. In: *PADL*, Springer, pp. 174–186, doi:10.1007/978-3-540-30557-6_14.

[6] Rachel Ben-Eliyahu & Rina Dechter (1994): *Propositional semantics for disjunctive logic programs*. *Annals of Mathematics and Artificial intelligence* 12, pp. 53–87, doi:10.1007/BF01530761.

[7] Claude Berge (1984): *Hypergraphs: combinatorics of finite sets*. 45, Elsevier.

[8] René van Bevern & Pavel V Smirnov (2020): *Optimal-size problem kernels for d-hitting set in linear time and space*. Information Processing Letters 163, p. 105998, doi:10.1016/j.ipl.2020.105998.

[9] Johann Birnick, Thomas Bläsius, Tobias Friedrich, Felix Naumann, Thorsten Papenbrock & Martin Schirneck (2020): *Hitting set enumeration with partial information for unique column combination discovery*. Proceedings of the VLDB Endowment 13(12), pp. 2270–2283.

[10] Thomas Bläsius, Tobias Friedrich, David Stangl & Christopher Weyand (2022): *An Efficient Branch-and-Bound Solver for Hitting Set*. In: ALENEX, SIAM, pp. 209–220, doi:10.1137/1.9781611977042.17.

[11] Endre Boros, K Elbassioni, Vladimir Gurvich & Leonid Khachiyan (2003): *An efficient implementation of a quasi-polynomial algorithm for generating hypergraph transversals*. In: ESA, Springer, pp. 556–567, doi:10.1007/978-3-540-39658-1_51.

[12] Endre Boros, Khaled Elbassioni, Vladimir Gurvich, Leonid Khachiyan & Kazuhisa Makino (2002): *Dual-bounded generating problems: All minimal integer solutions for a monotone system of linear inequalities*. SIAM Journal on Computing 31(5), pp. 1624–1643, doi:10.1137/S0097539701388768.

[13] Endre Boros, Vladimir Gurvich, Leonid Khachiyan & Kazuhisa Makino (2003): *On maximal frequent and minimal infrequent sets in binary matrices*. Annals of Math. and Artificial Intelligence 39, pp. 211–221, doi:10.1023/A:1024605820527.

[14] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca & Torsten Schaub (2020): *ASP-Core-2 input language format*. TPLP 20(2), pp. 294–309, doi:10.1017/S1471068419000450.

[15] Karthekeyan Chandrasekaran, Richard Karp, Erick Moreno-Centeno & Santosh Vempala (2011): *Algorithms for implicit hitting set problems*. In: SODA, SIAM, pp. 614–629, doi:10.1137/1.9781611973082.48.

[16] Keith L Clark (1978): *Negation as failure*. Logic and data bases, pp. 293–322, doi:10.1007/978-1-4684-3384-5_11.

[17] Guozhu Dong & Jinyan Li (2005): *Mining border descriptions of emerging patterns from dataset pairs*. Knowledge and Information Systems 8, pp. 178–202, doi:10.1007/s10115-004-0178-1.

[18] Dheeru Dua & Casey Graff (2017): *UCI Machine Learning Repository*. Available at `http://archive.ics.uci.edu/ml`.

[19] Niklas Eén & Niklas Sörensson (2003): *An extensible SAT-solver*. In: SAT, Springer, pp. 502–518, doi:10.1007/978-3-540-24605-3_37.

[20] Thomas Eiter & Georg Gottlob (1995): *On the computational cost of disjunctive logic programming: Propositional case*. Annals of Mathematics and Artificial Intelligence 15, pp. 289–323, doi:10.1007/BF01536399.

[21] Esra Erdem, Michael Gelfond & Nicola Leone (2016): *Applications of answer set programming*. AI Magazine 37(3), pp. 53–68, doi:10.1609/aimag.v37i3.2678.

[22] Michael L Fredman & Leonid Khachiyan (1996): *On the complexity of dualization of monotone disjunctive normal forms*. Journal of Algorithms 21(3), pp. 618–628, doi:10.1006/jagm.1996.0062.

[23] Dror Fried, Alexander Nadel & Yogev Shalmon (2023): *AllSAT for combinational circuits*. In: SAT, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 9:1–9:18, doi:10.4230/LIPIcs.SAT.2023.9.

[24] Martin Gebser, Benjamin Kaufmann & Torsten Schaub (2009): *Solution enumeration for projected Boolean search problems*. In: CPAIOR, Springer, pp. 71–86, doi:10.1007/978-3-642-01929-6_7.

[25] Martin Gebser, Benjamin Kaufmann & Torsten Schaub (2012): *Conflict-driven answer set solving: From theory to practice*. Artificial Intelligence 187, pp. 52–89, doi:10.1016/j.artint.2012.04.001.

[26] Martin Gebser, Benjamin Kaufmann & Torsten Schaub (2013): *Advanced conflict-driven disjunctive answer set solving*. In: IJCAI, AAAI Press.

[27] Michael Gelfond & Vladimir Lifschitz (1991): *Classical negation in logic programs and disjunctive databases*. New generation computing 9, pp. 365–385, doi:10.1007/BF03037169.

[28] Russell Greiner, Barbara A Smith & Ralph W Wilkerson (1989): *A correction to the algorithm in Reiter's theory of diagnosis*. Artificial Intelligence 41(1), pp. 79–88, doi:10.1016/0004-3702(89)90079-9.

[29] Dimitrios Gunopulos, Heikki Mannila, Roni Khardon & Hannu Toivonen (1997): *Data mining, hypergraph transversals, and machine learning*. In: *PODS*, pp. 209–216, doi:10.1145/263661.263684.

[30] Carito Guziolowski, Santiago Videla, Federica Eduati, Sven Thiele, Thomas Cokelaer, Anne Siegel & Julio Saez-Rodriguez (2013): *Exhaustively characterizing feasible logic models of a signaling network using answer set programming*. *Bioinformatics* 29(18), pp. 2320–2326, doi:10.1093/bioinformatics/btt393.

[31] Oliver Hädicke & Steffen Klamt (2011): *Computing complex metabolic intervention strategies using constrained minimal cut sets*. *Metabolic engineering* 13(2), pp. 204–213, doi:10.1016/j.ymben.2010.12.004.

[32] Céline Hébert, Alain Bretto & Bruno Crémilleux (2007): *A data mining formalization to improve hypergraph minimal transversal computation*. *Fundamenta Informaticae* 80(4), pp. 415–433.

[33] Alexey Ignatiev, Edward Lam, Peter J Stuckey & Joao Marques-Silva (2021): *A scalable two stage approach to computing optimal decision sets*. In: *AAAI*, 35, pp. 3806–3814, doi:10.1609/aaai.v35i5.16498.

[34] Alexey Ignatiev, Antonio Morgado & Joao Marques-Silva (2018): *PySAT: A Python Toolkit for Prototyping with SAT Oracles*. In: *SAT*, pp. 428–437, doi:10.1007/978-3-319-94144-8_26.

[35] Mohimenul Kabir (2024): *Minimal model counting via knowledge compilation*. *arXiv preprint arXiv:2409.10170*, doi:10.48550/arXiv.2409.10170.

[36] Mohimenul Kabir, Supratik Chakraborty & Kuldeep S Meel (2024): *Exact ASP counting with compact encodings*. In: *AAAI*, 38, pp. 10571–10580, doi:10.1609/aaai.v38i9.28927.

[37] Mohimenul Kabir, Flavio O Everardo, Ankit K Shukla, Markus Hecher, Johannes Klaus Fichte & Kuldeep S Meel (2022): *ApproxASP–a scalable approximate answer set counter*. In: *AAAI*, 36, pp. 5755–5764, doi:10.1609/aaai.v36i5.20518.

[38] Mohimenul Kabir & Kuldeep S Meel (2023): *A Fast and Accurate ASP Counting Based Network Reliability Estimator.* In: *LPAR*, 94, pp. 270–287, doi:10.29007/kc6q.

[39] Mohimenul Kabir & Kuldeep S Meel (2024): *On Lower Bounding Minimal Model Count*. *TPLP* 24(4), pp. 586–605, doi:10.1017/S147106842400036X.

[40] Mohimenul Kabir & Kuldeep S Meel (2025): *An ASP-Based Framework for MUSes*. *arXiv preprint arXiv:2507.03929 (to appear ICLP 2025 Technical Communications)*, doi:10.48550/arXiv.2507.03929.

[41] Mohimenul Kabir, Van-Giang Trinh, Samuel Pastva & Kuldeep S Meel (2025): *Scalable Counting of Minimal Trap Spaces and Fixed Points in Boolean Networks*. *arXiv preprint arXiv:2506.06013 (to appear CP 2025)*, doi:10.48550/arXiv.2506.06013.

[42] Richard M Karp (2009): *Reducibility among combinatorial problems*. In: *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, Springer, pp. 219–241, doi:10.1007/978-3-540-68279-0_8.

[43] Dimitris Kavvadias & Elias Stavropoulos (2005): *An efficient algorithm for the transversal hypergraph generation*. *Journal of Graph Algorithms and Applications* 9(2), pp. 239–264, doi:10.7155/jgaa.00107.

[44] Steffen Klamt & Ernst Dieter Gilles (2004): *Minimal cut sets in biochemical reaction networks*. *Bioinformatics* 20(2), pp. 226–234, doi:10.1093/bioinformatics/btg395.

[45] Tolga Kurtoglu, Sriram Narasimhan, Scott Poll, David Garcia, Lukas Kuhn, Johan de Kleer, Arjan van Gemund & Alexander Feldman (2009): *First international diagnosis competition-DXC'09*. *DX* 9, pp. 383–396.

[46] Anna L.D. Latour, Arunabha Sen & Kuldeep S. Meel (2023): *Solving the Identifying Code Set Problem with Grouped Independent Support*. In: *IJCAI*, pp. 1971–1978, doi:10.24963/ijcai.2023/219.

[47] Joohyung Lee & Vladimir Lifschitz (2003): *Loop formulas for disjunctive logic programs*. In: *ICLP*, Springer, pp. 451–465, doi:10.1007/978-3-540-24599-5_31.

[48] Charles E Leiserson, Marc Moreno Maza, Liyun Li & Yuzhen Xie (2010): *Parallel computation of the minimal elements of a poset*. In: *International Workshop on Parallel and Symbolic Computation*, pp. 53–62, doi:10.1145/1837210.1837221.

[49] Lin Li & Jiang Yunfei (2002): *Computing minimal hitting sets with genetic algorithm*. Algorithmica 32(1), pp. 95–106.

[50] Zhang Li, Wang Yisong, Xie Zhongtao & Feng Renyan (2021): *Computing Propositional Minimal Models: MiniSAT-Based Approaches*. Journal of Computer Research and Development 58(11), pp. 2515–2523.

[51] Yuliya Lierler (2005): *Cmodels–SAT-based disjunctive answer set solver*. In: LPNMR, Springer, pp. 447–451, doi:10.1007/11546207_44.

[52] Victor W Marek & Miroslaw Truszczyński (1999): *Stable models and an alternative logic programming paradigm*. In: The Logic Programming Paradigm, Springer, pp. 375–398.

[53] Wiktor Marek & Mirosław Truszczyński (1991): *Autoepistemic logic*. Journal of the ACM (JACM) 38(3), pp. 587–618, doi:10.1145/116825.116836.

[54] Gabriele Masina, Giuseppe Spallitta & Roberto Sebastiani (2023): *On CNF Conversion for Disjoint SAT Enumeration*. In: SAT, pp. 15:1–15:16, doi:10.4230/LIPIcs.SAT.2023.15.

[55] Gary McGuire, Bastian Tugemann & Gilles Civario (2014): *There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem via hitting set enumeration*. Experimental Mathematics 23(2), pp. 190–217, doi:10.1080/10586458.2013.870056.

[56] Carlos Mencıa, Alessandro Previti & Joao Marques-Silva (2015): *Literal-based MCS extraction*. In: IJCAI, 15, pp. 1973–1979.

[57] Keisuke Murakami & Takeaki Uno (2013): *Efficient algorithms for dualizing large-scale hypergraphs*. In: ALENEX, SIAM, pp. 1–13, doi:10.1137/1.9781611972931.1.

[58] James A Reggia, Dana S Nau & Pearl Y Wang (1983): *Diagnostic expert systems based on a set covering model*. International Journal of Man-Machine Studies 19(5), pp. 437–460, doi:10.1016/S0020-7373(83)80065-0.

[59] Raymond Reiter (1987): *A theory of diagnosis from first principles*. Artificial intelligence 32(1), pp. 57–95, doi:10.1016/0004-3702(87)90062-2.

[60] Ken Satoh & Takeaki Uno (2006): *Enumerating minimal explanations by minimal hitting set computation*. In: KSEM, Springer, pp. 354–365, doi:10.1007/11811220_30.

[61] Takahisa Toda (2013): *Hypergraph transversal computation with binary decision diagrams*. In: SEA, Springer, pp. 91–102, doi:10.1007/978-3-642-38527-8_10.

[62] Cong T Trinh, Aaron Wlaschin & Friedrich Srienc (2009): *Elementary mode analysis: a useful metabolic pathway analysis tool for characterizing cellular metabolism*. Applied Microbiology and Biotechnology 81, pp. 813–826, doi:10.1007/s00253-008-1770-1.

[63] Mirosław Truszczyński (2011): *Trichotomy and dichotomy results on the complexity of reasoning with disjunctive logic programs*. TPLP 11(6), pp. 881–904, doi:10.1017/S1471068410000463.

[64] Paola Vera-Licona, Eric Bonnet, Emmanuel Barillot & Andrei Zinovyev (2013): *OCSANA: optimal combinations of interventions from network analysis*. Bioinformatics 29(12), pp. 1571–1573, doi:10.1093/bioinformatics/btt195.

[65] Staal Vinterbo & Aleksander Øhrn (2000): *Minimal approximate hitting sets and rule templates*. International Journal of Approximate Reasoning 25(2), pp. 123–143, doi:10.1016/S0888-613X(00)00051-7.

[66] Franz Wotawa & David Kaufmann (2022): *Model-based reasoning using answer set programming*. Applied Intelligence 52(15), pp. 16993–17011, doi:10.1007/s10489-022-03272-2.

[67] Ionela Zevedei-Oancea & Stefan Schuster (2005): *A theoretical framework for detecting signal transfer routes in signalling networks*. Computers & Chemical Engineering 29(3), pp. 597–617, doi:10.1016/j.compchemeng.2004.08.026.

Logic Programming with Extensible Types

Ivan Perez
KBR @ NASA Ames Research Center
California, USA
ivan.perezdominguez@nasa.gov

Angel Herranz
Universidad Politecnica de Madrid
Madrid, Spain
angel.herranz@upm.es

Logic programming languages present clear advantages in terms of declarativeness and conciseness. However, the ideas of logic programming have been met with resistance in other programming communities, and have not generally been adopted by other paradigms and languages. This paper proposes a novel way to incorporate logic programming in an existing codebase in a typed functional programming language. Our approach integrates with the host language without sacrificing static typing, and leverages strengths of typed functional programming such as polymorphism and higher-order. We do so by combining three ideas. First, we use the extensible types technique to allow values of the host language to contain logic variables. Second, we implement a unification algorithm that works for any data structure that supports certain operations. Third, we introduce a domain-specific language to define and query predicates. We demonstrate our proposal via a series of examples, and provide aids to make the notation convenient for users, showing that the proposed approach is not just technically possible but also practical. Our ideas have been implemented in the language Haskell with very good results.

1 Introduction

Definitions in imperative and functional languages are structured around functions and procedures, which are executed by providing the inputs and evaluating the result. For example, for the Haskell type `data Nat = Zero | Suc Nat`, we can define addition as:

```
plus :: Nat -> Nat -> Nat
plus Zero    y = y
plus (Suc x) y = Suc (plus x y)
```

Although, at an abstract level, functions are relations between sets, it is not usually possible to treat functions as relations in a language like Haskell. For example, we cannot use `plus` to efficiently calculate the subtraction function or to generally calculate all tuples of inputs and outputs in the `plus` relation. Contrast this limitation with how one would write an analogous predicate in a logic programming language like Prolog:

```
plus(zero,   Y, Y).
plus(suc(X), Y, suc(Z)) :- plus(X,Y,Z).
```

In Prolog, `plus(A,B,C)` is true if `C` represents the sum of `A` and `B`. The predicate can be used to add numbers, subtract a number from another, or check if two numbers add up to a given third, or obtain tuples of inputs and output for which the relation holds: [1]

[1] We manually stop the production of solutions to the last query.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 248–262, doi:10.4204/EPTCS.439.18

```
?- plus(suc(suc(zero)),B,suc(suc(suc(zero)))).
B = suc(zero).
?- plus(zero,zero,suc(zero)).
false.
?- plus(A,suc(zero),C).
A = zero, C = suc(zero) ;
A = suc(zero), C = suc(suc(zero)) .
```

Integrating ideas from logic programming into other languages generally requires substantial changes to existing codebases and the types used throughout. Depending on the approach and the engine used to process queries, it may also lead to loss of static type safety or other features of the host language (e.g., polymorphism, higher order).

This paper describes a technique for integrating predicates in the style of logic programming in an existing codebase in a different programming language. We use Haskell to demonstrate our proposal, but the ideas can be applied to other languages. We show, with examples, that our approach requires very little work on the side of the programmer, can capture many of the use cases of logic programming languages, and can be enabled by convenient notation. Specifically, the contributions of this paper are:

- We present an interface for logic programming that facilitates introducing logic variables in algebraic datatypes and expressing unification constraints (Section 3).
- We show that the proposed approach is applicable to polymorphic types, and enables leveraging the host language's mechanisms for type inference and higher-order to implement type-safe higher-order logic programming (Section 4).
- We extend the language with *cuts*, which allow users to increase performance, provide determinism, and encode negation as failure (Section 5).

Section 6 discusses an implementation that demonstrates our proposal, Section 7 details related work, and Section 8 proposes future work.

2 Background

Extensible types [24] are a design pattern in which a data type is parameterized by a type function that is applied to every element of the definition. For example, given a type representing expressions, like `data Expr = Const Double | Add Expr Expr | Neg Expr`, we define the matching extensible type:

```
data ExprF f = ConstF (f Double)
             | AddF (f (ExprF f)) (f (ExprF f))
             | NegF (f (ExprF f))
```

If we use the polymorphic type `Identity` as type function `f`, the resulting representation is isomorphic to the original `Expr`. Other parametric types and type functions render different results. For example, a type that pairs elements with a tuple of `Int`s can be used to annotate values with the line and column where they were found in an input file, useful in compilers to report error information. Applying `Maybe` or `Either`, an extensible type makes every element optional, a representation that is useful in parsing to mark branches of an abstract syntax tree

(AST) that failed to parse. Other type functions enable changing type definitions to introduce new cases, prune branches, replace elements, etc. The composition of extensible types can capture language embeddings.

The application of a specific type function to an extensible type does not determine how it should be interpreted. For example, `Either String` can be used to annotate failed AST branches with the reasons why values could not be parsed from an input file, but also to replace branches in the AST by variables with the given variable names. This idea will be used in future sections to replace portions of a datatype with variables in predicate definitions and logic programming queries.

In the rest of the text, we refer to types that are parameterized in this manner simply as *extensible types*. Other approaches to parameterize a type by a type function used in its definition are further discussed in Section 7.

3 Logic Programming with Extensible Types

This section introduces primitives to define and combine predicates, and ways to capture relations between values of algebraic datatypes. We first introduce basic types, and simple primitives and connectives. We later show how to replace portions of values with logic variables, and how to express relations involving types with variables.

3.1 Goals, Primitives and Boolean Combinators

The elementary type in our proposal is a `Goal`, which denotes a logic goal or, put simply, something that must be proven. We keep the type abstract for now and discuss implementation details later. To interact with `Goals`, we provide the function `repl :: Goal -> IO ()` that, when applied to a `Goal`, produces possible solutions one by one, similar to the REPL of a logic programming language. If constraints apply for the goal to hold, `repl` prints the constraints; otherwise, it prints "`true.`" or "`false.`". In this paper, we align queries to and results from `repl` for readability.

3.1.1 Primitives

We provide `succeed :: Goal`, which holds without additional constraints, and its counterpart, `fail :: Goal`, which always fails. We can evaluate either goal in a session with the `repl` function, as follows:

```
> repl fail
false.
```

3.1.2 Boolean Connectives

Our counterparts for the boolean connectives *and* and *or*, which we denote `(@@)` and `(@|)`, allow users to combine goals:

```
(@@) :: Goal -> Goal -> Goal
(@|) :: Goal -> Goal -> Goal
```

Example The following queries show how we can use the boolean connectives to combine `succeed` and `fail`. The results should be straightforward:

```
> repl (succeed @@ fail)
false.
> repl (succeed @@ (fail @| succeed))
true.
```

3.2 Terms and Logic Variables

To introduce logic variables in values, we apply a type function to extensible types. We introduce a custom sum type `Term`, which can represent a logic variable with a name, or an actual value of a given type:

```
data Term a = Var String | Compound a
```

When `Term` is applied to an extensible type, every element inside the latter can potentially be replaced with a logic variable, allowing us to describe values in which some portions are concrete and some portions are not.

Example Given the usual encoding of Peano numbers using a data type defined as `data Nat = Zero | Suc Nat`, the equivalent extensible type in Haskell would be:

```
data NatF f = ZeroF | SucF (f (NatF f))
```

We can use `NatF Term` to represent a natural number where part of the definition is substituted by a variable. To make the complete number replaceable with a variable, we enclose the type inside an additional `Term`:

```
type NatTerm = Term (NatF Term)
```

Examples of values of type `NatTerm` include `Var "y"`, representing a natural number denoted by the variable `"y"`, `Compound (SucF (Var "x"))`, representing the successor of `"x"`, and `Compound (SucF (Compound ZeroF))`, representing 1. In Prolog, such terms could be encoded as `Y`, `suc(X)`, and `suc(zero)`, respectively.

3.3 Term Unification

We have designed a domain-specific language (DSL) to write predicates on types for which we can perform term unification and variable substitution. To unify two terms, we provide `(===) :: Term a -> Term a -> Goal`.[2]

Example We can define the successor predicate as:[3]

```
isSuc :: NatTerm -> NatTerm -> Goal
isSuc x y = Compound (SucF x) === y
```

[2]The function is not fully polymorphic; we detail constraints applicable to the parameter `a` in Section 6.

[3]Compare with the Prolog program `is_suc(X,Y) :- suc(X) = Y.`

We now query this predicate in a session to check if it holds for two *ground* values:[4]

```
> repl $ isSuc (Compound (SucF (Compound ZeroF)))
               (Compound (SucF (Compound (SucF (Compound ZeroF)))))
true.
> repl $ isSuc (Compound (SucF (Compound ZeroF)))
               (Compound (SucF (Compound ZeroF)))
false.
```

The real power of our approach is that we can now use variables to provide one value and "obtain" the other, or rather, the *answer substitutions*:

```
> repl $ isSuc (Compound (SucF (Compound ZeroF))) (Var "x")
x = Compound (SucF (Compound (SucF (Compound ZeroF)))).
> repl $ isSuc (Var "x") (Compound (SucF (Compound ZeroF)))
x = Compound ZeroF.
```

3.4 Existential Quantification

The scope of variables as presented so far is, by default, global. A variable with a fixed name being used inside a function will be considered to be the same as a variable with the same name (and type) used elsewhere in the same query.

To introduce free variables in the *body* of predicates, akin to introducing free variables in the antecedent in predicate definitions in logic programming languages, we define the function `exist :: (Term a -> Goal) -> Goal`. When the function `exists` is applied to an argument predicate, it ensures that the variable provided to the given predicate is free. To avoid name clashes, we recommend that users always introduce variables with `exists`.

Example Using all the definitions provided so far, we can implement the predicate `leq` (i.e., less than or equal to) to compare two natural numbers, `x` and `y`. The first rule of the comparison is that, if the first number `x` is zero, then `leq x y` must necessarily hold as there is no smaller number, that is, `x === Compound ZeroF`. The second rule is that, if both elements are successors of other elements, respectively `x'` and `y'`, then the goal holds if it holds for `x'` and `y'`. Combining both rules we obtain:

```
leq :: NatTerm -> NatTerm -> Goal
leq x y = x === Compound ZeroF
       @| ( exists $ \x' -> exists $ \y' ->
            x === Compound (SucF x') @@ y === Compound (SucF y') @@ leq x' y' )
```

Introducing free variables at top of a definition helps group rules and aids readability:

```
leq :: NatTerm -> NatTerm -> Goal
leq x y = exists $ \x' -> exists $ \y' ->
     x === Compound ZeroF
  @| x === Compound (SucF x') @@ y === Compound (SucF y') @@ leq x' y'
```

As illustrated above, predicates can be *recursive*.

[4] We use *ground* to refer to *terms* that do not contain variables. The word ground also has meaning when discussing data types and generic programming, but we use the word exclusively with the former meaning.

3.5 Notation

The ideas of our proposal are applicable to other languages, but Haskell's ability to overload notation and define operators can make logic programming more convenient. Specifically, Haskell allows us to define new operators and adjust their associativities and priorities, making `(@@)` bind more strongly than `(@|)`, and `(===)` bind more strongly than either of them. We define synonyms `C` and `V` for, respectively, `Compound` and `Var`, and pattern synonyms `Zero = C ZeroF` and `Suc x y = C (SucF x y)`. Using these facilities, `leq` can now be defined more succinctly as follows:

```
leq :: NatTerm -> NatTerm -> Goal
leq x y = exists $ \x' -> exists $ \y' ->
     x === Zero
  @| x === Suc x' @@ y === Suc y' @@ leq x' y'
```

To help understand the technical details of our approach, we refrain from relying too heavily on syntactic sugar during this exposition. Our implementation provides aids to make using logic programming more convenient, which we discuss in Section 6.

4 Polymorphism and Higher Order

The ability to write predicates using the approach described so far extends also to polymorphic types. Let us demonstrate with the type of polymorphic lists, frequently used in logic and functional programs. The standard list type definition, `data List a = Nil | Cons a (List a)`, can be extended with an extra type function as follows:

```
data ListF f a = NilF | ConsF (f a) (f (ListF f a))
```

By applying the type function `Term` to `ListF`, we can use logic variables in place of elements of the list, or the tail of the list at any give point:

```
type ListTerm a = Term (ListF Term a)
```

Example We can combine `ListTerm` with `NatF` to represent lists of natural numbers:

```
type NatListTerm = ListTerm (NatF Term)
```

The following, for now, rather verbose term encodes the list `[0, 1, 2]`:

```
l1 :: NatListTerm
l1 = C $ ConsF (C ZeroF)
   $ C $ ConsF (C (SucF (C ZeroF)))
   $ C $ ConsF (C (SucF (C (SucF (C ZeroF))))) Nil
```

The following term encodes the list that starts with a 1, and whose tail is represented by a variable `"tl"`:

```
l2 :: NatListTerm
l2 = C (ConsF (C (Suc (C Zero))) (V "tl"))
```

Notation Like before, we introduce pattern synonyms to simplify writing terms of type `ListTerm`: `Cons x y = C (ConsF x y)` and `Nil = C NilF`. Using all pattern synonyms defined so far, the list `l1` in the previous example can be defined as:

```
l1 :: NatListTerm
l1 = Cons Zero $ Cons (Suc Zero) $ Cons (Suc (Suc Zero)) Nil
```

Our implementation allows us to write expressions like `[0, "x"]` to mean a list with a first element 0 and a second element being the variable `"x"`. We delay notation aids to Section 6, to help the reader gain intuition about how our approach works.

4.1 Polymorphism and Type Safety

Using extensible types does not prevent the host programming language from performing type checking, including for types whose non-extensible variants were polymorphic. If we try to use a term with the wrong type in the definition of a predicate or in a unification constraint, the type checker detects it just like it would any other type error.

Example Take the following predicate that checks if an element is the head of a list:

```
isHead :: ListTerm a -> Term a -> Goal
isHead x y = exists $ \tl -> x === Cons y tl
```

If we call `isHead` with a second argument of the wrong type, the Haskell compiler's type checker warn:

```
> repl $ isHead l1 ((Var "x") :: Term Bool)
<interactive>:9:19: error:
      Couldn't match type 'Bool' with 'NatF Term'
      Expected type: Term (NatF Term)
        Actual type: Term Bool
```

This level of safety is especially important when using free variables. A host language with strong, static types can ensure that we are using variables in type-consistent ways.

Example We can check if a value is in a list by checking against the head or recursing into the tail. There is no rule for the empty list, which makes the goal fail in that case:

```
member :: Term a -> ListTerm a -> Goal
member x xs =
     ( exists $ \tl -> xs === Cons x tl )
  @| ( exists $ \hd -> exists $ \tl -> xs === Cons hd tl @@ member x tl )
```

The variable `hd` in the second rule has type `Term a`, and `tl` has type `List a`. The compiler can infer this because both are arguments to `Cons` and the resulting term unifies with `xs`, whose type is known. If, for example, we introduce a condition `xs === hd`, or `member tl tl`, the compiler will detect that we are using variables in inconsistent ways.

Let us further demonstrate the type safety features of our approach with a predicate that checks if a list is sorted. The predicate holds trivially for lists of zero or one elements; if there are more, we check the first two and recurse into the tail of the list.

```
sorted v =
     v === Nil
  @| ( exists $ \e1 -> v === Cons e1 Nil)
  @| ( exists $ \e1 -> exists $ \e2 -> exists $ \ts ->
            v === Cons e1 (Cons e2 ts)
         @@ leq e1 e2
         @@ sorted (Cons e2 ts) )
```

In this case, the compiler infers that `v` has type `NatList`, which is defined as `ListTerm (NatF Term)` and expands to `Term (ListF Term (NatF Term))`.

4.2 Higher-order Logic Programming

Haskell's support for first-class functions immediately empowers our approach with higher-order, allowing us to pass predicates as arguments to other predicates.

Example Let us illustrate with a generalized version of `sorted` that takes a comparison *predicate* as argument:

```
sortedWith :: (Term a -> Term a -> Goal) -> ListTerm a -> Goal
sortedWith compare v =
     v === Nil
  @| ( exists $ \x -> v === Cons x Nil )
  @| ( exists $ \x1 -> exists $ \x2 -> exists $ \xs ->
            v === Cons x1 (Cons x2 xs)
         @@ compare x1 x2
         @@ sortedWith compare (Cons x2 xs) )
```

Because Haskell is strongly and statically typed, it provides a level of safety that surpasses what most implementations of Prolog offer,[5] since they cannot assure that types match without added, hand-coded runtime checks. In general, calling a Prolog predicate with arguments of the wrong types may return an incorrect result, making this kind of type error hard to identify. Providing arguments of the wrong type may return `false`, just as if the predicate did not hold for those inputs (because it does not!), but may also incorrectly return `true` (e.g., `append([],1,1)` is `true` even though `1` is not a list).

We can use the same approach to generalize functions and turn them into predicates, such as the standard function `map` that applies a transformation to each element in a list, as illustrated by the following predicate `mapP`:

```
mapP :: (Term a -> Term b -> Goal) -> ListTerm a -> ListTerm b -> Goal
mapP f l1 l2 =
     l1 === Nil @@ l2 === Nil
  @| ( exists $ \l10 -> exists $ \l1s -> exists $ \l20 -> exists $ \l2s ->
            l1 === Cons l10 l1s
         @@ l2 === Cons l20 l2s
         @@ f l10 l20 @@ mapP f l1s l2s )
```

[5] Some Prolog implementations, like Ciao Prolog, support static analysis via compile-time assertions.

Given a predicate `isSuc`, which pairs each number with its successor, we can use it to add 1 to every element of a list using `mapP` as follows:

```
listPlusOne :: ListTerm (NatF Term) -> ListTerm (NatF Term) -> Goal
listPlusOne = mapP isSuc
```

Haskell's type checker prevents runtime errors by ensuring, at compile time, that `isSuc` has type `Term (NatF Term) -> Term (NatF Term) -> Goal`.

5 Cuts

In logic programming, *cuts* limit the use of backtracking to search for alternative solutions. Consider the following (incorrect) implementation of the remainder algorithm:

```
remainder :: NatTerm -> NatTerm -> NatTerm -> Goal
remainder n q r = lt n q @@ n === r
              @| exists $ \diff -> plus q diff n @@ remainder diff q r
```

This definition does not work if `q` is zero, since the first rule fails, and the execution of the second rule leads the program into an infinite loop. Unfortunately, adding a protection rule like `q === Zero @@ fail` does not help, since, if `q` is zero, that rule fails and the evaluation backtracks, eventually falling into the last rule again.

To prevent such cases, we introduce the functions `scope` and `(@!)`, which help control backtracking. Inspired by the notion of cuts in Prolog, we refer to `(@!)` as our own cut operator. We re-write `remainder` using these two functions as follows:

```
remainder :: NatTerm -> NatTerm -> NatTerm -> Goal
remainder n q r =
  scope $ q === Zero @! fail
       @| lt n q @@ n === r
       @| (exists $ \diff -> plus q diff n @@ remainder diff q r)
```

Callers to `remainder` are unaware that the predicate is implemented using cuts, provided that uses of `(@!)` are delimited by `scope`, limiting how far the cut applies. Without `scope` in the definition of `remainder`, predicates using `remainder` in a rule would see alternative (i.e., `(@|)`) rules being skipped over if `remainder` fails due to `q` being `Zero`.

Negation as Failure Cuts can be used to implement a form of negation, with:

```
neg :: Goal -> Goal
neg p = scope $ p @! fail @| succeed
```

Example It is frequently useful to state that two terms cannot unify, for which we define an operator `(=/=)` as:

```
(=/=) :: Term a -> Term a -> Goal
(=/=) x y = neg (x === y)
```

Similarly, we can implement a predicate that holds only if a given term is *not* a member of a given list:

```
notMember :: Term a -> List a -> Goal
notMember x xs = neg (member x xs)
```

This kind of negation, called *negation as failure*, is a weak form of negation. If either of the arguments of `(=/=)` is still a variable at the time when the unification algorithm tries to evaluate whether the goal holds, the unification `x === y` will hold, making its negation fail. This limits the usefulness of this form of negation. For example, one cannot use `notMember` as defined above to find possible values of a variable that are not members of a list. Instead, it is necessary to *ground* the term first. To this end, we provide the predicate `isGround` which, for any type for which ground terms have a finite representation, holds only if the given argument contains no variables.

6 Implementation

We have implemented the ideas in this paper in Haskell[6], including types representing `Terms` and `Goals`; goal building functions and combinators; classes that define the operations that types must support for unification to be used on them; a unification algorithm; and execution functions to evaluate goals. Overall, our implementation only needs 350 lines of code, without considering spaces or comments. We have implemented this solution with the aim of demonstrating the capabilities explained, and explore design decisions, syntax and embeddings. Evaluating the performance of the solution and comparing it with existing logic programming implementations is out of the scope of this paper and considered future work.

High-level Description Our implementation defines two key types: a polymorphic type `Term`, described in Section 3, and a type `Goal`, which represents a goal. To perform unification for terms of arbitrary types, we require that three operations be supported on `Terms`: the ability to unify two terms of a specific type, the ability to check if a variable is used in a term, and the ability to substitute a variable by a term inside another term. We capture these operations in Haskell via type classes, the key one being the type class `Logic`, which represents types for which the aforementioned operations are defined.

To evaluate goals, we provide several functions, including: `repl :: Goal -> IO ()`, which prints solutions one by one, letting users control the production of solutions with the keyboard, and `findAll :: Logic a =>Term a -> Goal -> [Term a]`, which provides all values for an argument variable `Term` under which a `Goal` may hold.

Our implementation represents goals using a tree-like structure with unification constraints in the nodes. An internal function `solve` traverses the tree, accumulating unification constraints, substituting variables with their expected values in other terms, and discarding branches that cannot be unified. The function `solve` produces all possible solutions, each of which assigns values to variable terms. Our function `repl` leverages Haskell's inherent laziness to produce and print solutions one by one.

Notation and Usability To make our DSL convenient, we have implemented facilities to make the notation succinct and familiar, and reduce how much code users must write. Our implementation uses generic programming [18] to generate instances of the classes that our

[6]Our implementation has been made publicly available at https://github.com/ivanperez-keera/telos

unification function requires to operate on algebraic data types. For example, for the type `NatF`, users need to declare several type class instances, but do not need to implement them manually. Our solution also facilitates writing terms of some common types. For example, Peano natural numbers are printed as `1`, `2`, etc. when they are ground terms, and a number plus a variable otherwise (e.g., `1 + x`, `55 + z`). We implement similar aids for `ListF`, so that values can be shown and given in a familiar notation (e.g., `1 : x : 5 : xs`). Where a `Term` is expected, a literal string is interpreted as variable term. Overall, this renders very concise expressions:

```
> repl (plus 1 "x" 5)
x = 4.
> repl (isTail [1, 2, 3] [2, 3])
true.
```

The same mechanisms for syntax overloading may not be available in other languages that otherwise support higher-kinded polymorphism or dynamically replacing a value by a sum type, meaning that the notation in such languages could be more cumbersome.

7 Related Work

Functional-Logic Programming Languages The creation of languages that integrate logic and functional programming using theoretical frameworks and efficient implementations has been subject to prior study [4, 6]. Languages in this category include Babel [19], K-LEAF [8], ALF [11], Curry [10], and Escher [17], which support a functional style, and Gödel [14], Mercury [28], and λProlog [21], which embrace a logic programming style. Curry, in particular, is strongly inspired by Haskell but incorporates logical variables and uses narrowing as operational semantics to compute the value of expressions with free variables [9]. Instead of creating a new language, our work shows how an existing functional language can be empowered with logic programming capabilities, without additional compiler extensions and without calling an external logic programming engine.

"Functions" in Logic Programming Languages Prolog includes limited higher-order capabilities like `call/N` and `apply/3` [22]. The Prolog implementation Ciao [12] allows, via its metaprogramming libraries, using predicates in a functional style, treating the last argument as the result of the function. In our case, functions and higher-order come built-in with the host language and are immediately exploitable by programmers. Furthermore, because we rely on a statically typed language, our approach provides a level of static safety.

Typed Logic Programming Languages Prolog has previously been extended with static types [20, 25, 3], but these extensions are not integrated in most widely used Prolog systems. The implementation Ciao Prolog [12] provides a mechanism of assertions based on *regular types* that are checked statically. In contrast, our proposal leverages the type system of the host programming language.

Logic Programming Embeddings Prior attempts at embedding logic programming in functional languages by [29], [5], [27] and [7] require adapting the types by hand to use them in

logical predicates. Work by [15] to embed relations in O'Caml require introducing *projections* and *injections* to move between terms and functional values. In contrast, our approach is applicable to arbitrary algebraic data types, with generic programming aids to facilitate operating with them, simplifying the process. The application of a systematic extension pattern, rather than hand-coded extensions, leads to regular and predictable ways to add variables to algebraic data types.

In terms of the implementation of goal evaluation, prior embeddings use an interpreter based on an evaluation monad [29, 5, 27, 26], make goals data *streams* and explicitly introduce a backtracking lazy stream monad in an otherwise strict setting [15], or use continuation passing style and exceptions to evaluate logic programs more efficiently [7]. In contrast, we use an internal representation of goals as a tree of unification constraints, parameterized by a counter used to generate free variables. Our encoding provides a more fine-grained control of the counter than what we could obtain by using a (lazy) state monad, and introduces a level of laziness that is crucial to generate solutions efficiently and implement the REPL. Another difference in implementation between miniKanren [15] and our work is that the former uses a type-unsound internal representation encapsulated behind a type-safe API. Our approach implements unification without sacrificing type safety, and term unification can only be applied to terms of the same type.

These embeddings also differ from our work in terms of expressiveness: for example, we only support equality constraints, and implement a limited form of disequality using a (weak) form of negation via cuts, whereas miniKanren and the work of [26] support both equality and disequality.

A more recent publication presents typedKanren [16], an embedding of a relational language based on miniKanren inside Haskell. Like our proposal, typedKanren enables writing strongly typed predicates in Haskell, and uses a type for term that is similar to ours (except for added strictness and the use of `Int` instead of `String` to represent variables). A key difference is that typedKanren requires `Logical` variants of types to be written for standard types (e.g., binary tries), whereas our representations are based on extensible types and are therefore more versatile. Like miniKanren, typedKanren supports disequalities, while we do not support them in the general case. Goals are monads in typedKanren, which results in a different style when specifying predicates, while goals in our language are not monads, and we instead provide additional constructs and syntactic aids to make the notation more convenient and closer to the syntax of Prolog.

Higher-kinded Type Parametrizations Our solution uses extensible types to parameterize types with type functions. [23] propose an alternative technique that uses *type families* and adds a parameter `f` to every branch of an algebraic data type's definition, as opposed to every element inside the type. The work of Najd et al. also allows introducing `Term`-like wrappers around all elements of the abstract data type. However, we find extensible types straightforward due to the minimal work required to replace *any* part of a type with a variable by applying `Term` to the extensible type.

The library `barbies` [1] implements generic mechanisms to work with types parameterized by a functor similar to extensible types, and Barbies-th [2] or Higgledy [13] could help generate higher-kinded types in Haskell. We have yet to investigate how to take advantage of such approaches.

8 Future Work

This paper has shown how to embed logic programming in a statically typed functional programming language. To that end, we used extensible types to replace any portion of an algebraic datatype with variables, and provided a mechanism to express unification constraints between values with variables. We further extended the language with boolean connectives, and ways to introduce free variables. We showed by example that we can leverage the host language's type inference and higher-order to make code reusable without sacrificing type safety. We closed our discussion with an overview of our implementation, and an evaluation of the differences with other approaches.

The approach proposed in this paper could be used to represent other kinds of constraints, allowing us to implement constraint-logic programming in Haskell. Introducing disequality constraints in our language would overcome the limitations of our current implementation of disequality using "negation as failure".

We are currently exploring how the language provided in this paper could be used to generate values that meet constraints in property-based testing, rather than first generating values and then filtering based on constraints (thus discarding many values).

Our experiments indicate that the semantics of our goal evaluation functions coincides with that of Prolog. We would like to carry out a more detailed and formal evaluation to compare our inference engine with Prolog's. Additionally, we have not discussed benchmarking our unification algorithm against existing implementations, as it is a topic that deserves careful and detailed evaluation, and we consider future work.

In future work, we would like to make our DSL more user-friendly and the notation closer to that of Prolog or other logic programming languages, without sacrificing the embedding in a host language. Prolog predicates are normally defined by multiple rules, where the head of the rule can unify with the arguments. In contrast, in our host language, pattern matching cannot be used to perform unification, so we need to use `(===)` in definitions to split the inputs into their component parts. We consider leveraging compiler extensions to mimic Prolog's unification of clause heads as future work.

In our proposal, users have to define an extensible type counterpart of the types they want to work with, wrap values into terms, and transform ground terms back into values of the original, non-extensible type. Although our notation aids partly simplify that process, we plan to investigate how template meta-programming could help generate the extensible type associated to a given algebraic datatype automatically.

Prolog allows using underscores in place of terms, to indicate that the value in place should unify, but not capture it, equivalent to introducing free variables where underscores are used. Our implementation supports wildcard patterns, or underscores, on the left-hand side of a predicate definition. In the future, we plan to investigate how to support wildcard patterns also in the right-hand side of predicate definitions.

Finally, we plan to evaluate how to remove the need to manually add `scope` to limit the effect of cuts in our language.

References

[1] *Barbies*. https://hackage.haskell.org/package/barbies.

[2] *barbies-th*. https://hackage.haskell.org/package/barbies-th.

[3] João Barbosa, Mário Florido & Vítor Santos Costa (2022): *Data Type Inference for Logic Programming*. In Emanuele De Angelis & Wim Vanhoof, editors: *Logic-Based Program Synthesis and Transformation*, Springer International Publishing, Cham, pp. 16–37, doi:10.1007/978-3-030-98869-2_2.

[4] Marco Bellia & Giorgio Levi (1986): *The relation between logic and functional languages: a survey*. *The Journal of Logic Programming* 3(3), pp. 217–236, doi:10.1016/0743-1066(86)90014-2.

[5] Koen Claessen & Peter Ljunglöf (2001): *Typed Logical Variables in Haskell*. In: *2000 ACM SIGPLAN Haskell Workshop (Satellite Event of PLI 2000)*.

[6] Doug DeGroot & Gary Lindstrom, editors (1986): *Logic Programming: Functions, Relations, and Equations*. Prentice-Hall.

[7] Conal Elliott & Frank Pfenning (1991): *A semi-functional implementation of a higher-order logic programming language*. *Topics in Advanced Language Implementation*, pp. 289–325.

[8] Elio Giovannetti, Giorgio Levi, Corrado Moiso & Catuscia Palamidessi (1991): *Kernel-LEAF: A logic plus functional language*. *Journal of Computer and System Sciences* 42(2), pp. 139–185, doi:10.1016/0022-0000(91)90009-T.

[9] M. Hanus (2013): *Functional Logic Programming: From Theory to Curry*. In: *Programming Logics - Essays in Memory of Harald Ganzinger*, Springer LNCS 7797, pp. 123–168, doi:10.1007/978-3-642-37651-1_6.

[10] M. Hanus, H. Kuchen & J.J. Moreno-Navarro (1995): *Curry: A Truly Functional Logic Language*. In: *Proc. ILPS'95 Workshop on Visions for the Future of Logic Programming*, pp. 95–107.

[11] Michael Hanus (1991): *The ALF system: An efficient implementation of a functional logic language*. In Harold Boley & Michael M. Richter, editors: *Processing Declarative Knowledge*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 414–416, doi:10.1007/BFb0013551.

[12] M. V. Hermenegildo, F. Bueno, M. Carro, P. López-García, E. Mera, J. F. Morales & G. Puebla (2012): *An overview of Ciao and its design philosophy*. *Theory and Practice of Logic Programming* 12(1–2), p. 219–252, doi:10.1017/S1471068411000457.

[13] *Higgledy*. https://hackage.haskell.org/package/higgledy.

[14] P. Hill & J.W. Lloyd (1994): *The Gödel Programming Language*. Logic programming, MIT Press.

[15] Dmitry Kosarev & Dmitry Boulytchev (2016): *Typed Embedding of a Relational Language in OCaml*. In: *Proceedings ML Family Workshop / OCaml Users and Developers workshops, ML/OCAML 2016, Nara, Japan, September 22-23, 2016*, pp. 1–22.

[16] Nikolai Kudasov & Artem Starikov (2024): *typedKanren: Statically Typed Relational Programming with Exhaustive Matching in Haskell*. arXiv preprint arXiv:2408.03170, doi:10.48550/arXiv.2408.03170. Presented at miniKanren 2024.

[17] John W Lloyd (1999): *Programming in an integrated functional and logic language*. *Journal of Functional and Logic Programming* 3(1-49), pp. 68–69.

[18] José Pedro Magalhães, Atze Dijkstra, Johan Jeuring & Andres Löh (2010): *A generic deriving mechanism for Haskell*. In: *Proceedings of the Third ACM Haskell Symposium on Haskell*, Haskell '10, New York, NY, USA, p. 37–48, doi:10.1145/1863523.1863529.

[19] Juan José Moreno-Navarro & Mario Rodríguez-Artalejo (1988): *BABEL: A Functional and Logic Programming Language based on Constructor Discipline and Narrowing*. In: *International Conference on Algebraic and Logic Programming*, doi:10.1007/3-540-50667-5_74.

[20] Alan Mycroft & Richard A. O'Keefe (1984): *A polymorphic type system for prolog*. *Artificial Intelligence* 23(3), pp. 295–307, doi:10.1016/0004-3702(84)90017-1.

[21] Gopalan Nadathur & Dale A. Miller (1986): *Higher-order logic programming*. In Ehud Shapiro, editor: *Third International Conference on Logic Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 448–462, doi:10.1007/3-540-16492-8_94.

[22] Lee Naish (1996): *Higher-order logic programming in Prolog*. In: *Proc. Workshop on Multi-Paradigm Logic Programming, JICSLP*, 96, pp. 1–23.

[23] Shayan Najd & Simon Peyton Jones (2017): *Trees that grow*. *Journal of Universal Computer Science* 21(1), pp. 42–62.

[24] Ivan Perez (2023): *Types that Change: The Extensible Type Design Pattern*. In: *Proceedings of the 1st ACM SIGPLAN International Workshop on Functional Software Architecture*, FUNARCH 2023, New York, NY, USA, p. 49–62, doi:10.1145/3609025.3609475.

[25] Tom Schrijvers, Vítor Santos Costa, Jan Wielemaker & Bart Demoen (2008): *Towards Typed Prolog*. In: *Proceedings of the 24th International Conference on Logic Programming*, ICLP '08, Springer-Verlag, Berlin, Heidelberg, p. 693–697, doi:10.1007/978-3-540-89982-2_59.

[26] Tom Schrijvers, Peter Stuckey & Philip Wadler (2009): *Monadic constraint programming*. *Journal of Functional Programming* 19(6), p. 663–697, doi:10.1017/S0956796809990086.

[27] Mehul Chandrakant Solanki (2012): *Embedding Programming Languages: Prolog in Haskell*. Master's thesis, University of Northern British Columbia.

[28] Zoltan Somogyi, Fergus Henderson & Thomas Conway (1996): *The execution algorithm of mercury, an efficient purely declarative logic programming language*. *The Journal of Logic Programming* 29(1), pp. 17–64, doi:10.1016/S0743-1066(96)00068-4.

[29] J Michael Spivey & Silvija Seres (1999): *Embedding Prolog in Haskell*. In: *Proceedings of Haskell*, 99, pp. 1999–28.

Fixpoint Semantics for DatalogMTL with Negation*

Samuele Pollaci
Vrije Universiteit Brussel
Brussels, Belgium
Katholieke Universiteit Leuven
Leuven, Belgium
Samuele.Pollaci@vub.be

DatalogMTL$^\neg$ is an extension of Datalog with metric temporal operators enriched with unstratifiable negation. In this paper, we define the stable, well-founded, Kripke-Kleene, and supported model semantics for DatalogMTL$^\neg$ in a very simple and straightforward way, by using the solid mathematical formalism of Approximation Fixpoint Theory (AFT). Moreover, we prove that the stable model semantics obtained via AFT coincides with the one defined in previous work, through the employment of pairs of interpretations stemming from the logic of here-and-there.

1 Introduction

Datalog [1] is a declarative query language that extends relational algebra with recursion. The last few decades have witnessed the emergence of several variants of Datalog and their implementations [18], aiming to solve data management tasks in various applications [16]. One of such variants is DatalogMTL [8], which extends the Datalog language with operators from metric temporal logic [21], and finds applications in ontology-based query answering [8, 17, 19, 20], stream reasoning [30, 31], and reasoning for the financial sector [23, 22, 9], amongst others.

For instance, we can use the following DatalogMTL rule to express that an adult has to wait at least 6 hours after the assumption of a paracetamol tablet, before taking another one:

$$\mathit{NoMoreParacetamol}(x) \leftarrow \mathit{Adult}(x) \wedge \diamondminus_{[0,6]}\, \mathit{TakesParacetamol}(x),$$

where $\diamondminus_{[0,6]}\, \mathit{TakesParacetamol}(x)$ is true at a timepoint t, if $\mathit{TakesParacetamol}(x)$ was true at some point in the last 6 hours, i.e. at some time t' in the interval $[t-6,t]$. Other than $\diamondminus_{[0,6]}$, there are a few more metric temporal operators that are allowed in the expressions of DatalogMTL. For instance, if φ is a formula and $[t_1,t_2]$ is an interval on the rational or integer timeline, then $\boxminus_{[t_1,t_2]}\varphi$ holds at time t if φ holds at all the timepoints of the interval $[t-t_2,t-t_1]$. Analogously, there are also the future-oriented counterparts of $\diamondminus$ and $\boxminus$, namely $\diamondplus$ and $\boxplus$, respectively: $\diamondplus_{[t_1,t_2]}\ \varphi$ holds at time t if φ holds at some timepoint of the interval $[t+t_1,t+t_2]$; and similarly for $\boxplus_{[t_1,t_2]}\varphi$. A DatalogMTL dataset is then a set of facts, each paired with a time interval. For example, if $\mathit{TakesParacetamol}(\mathit{John})@[8,8]$ is in a dataset, then it holds that *John* took a paracetamol tablet at 8a.m.

In 2021, DatalogMTL was extended with stratified negation as failure [10], allowing negated atoms in the body of rules, but no recursion. Recently, the range of capabilities of DatalogMTL was further expanded by equipping DatalogMTL with unstratifiable negation under the stable model semantics [29], while extending both DatalogMTL with stratified negation and Datalog with stable model negation. In such paper, to obtain the stable models of a dataset and a program, the authors employed interpretations

*This study was funded by Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G0B2221N)

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 263–277, doi:10.4204/EPTCS.439.19

© S. Pollaci
This work is licensed under the
Creative Commons Attribution License.

similar to the ones used for the logic of *here-and-there* (HT) [15, 24], namely pairs (I,J) of (two-valued) interpretations such that the atoms evaluated as *true* by I are *true* under J as well. Although used differently, this same framework with pairs of interpretations sits at the core of numerous applications of Approximation Fixpoint Theory (AFT) [12, 13], prompting the creation of this paper.

AFT was first developed to study the semantics of non-monotonic logics, such as logic programming, autoepistemic logic, and default logic. It has been successfully applied in several other areas, like abstract argumentation [26], active integrity constraints [5], stream reasoning [2], and constraint languages for the semantic web [6]. In the context of (consistent) AFT [13], a pair (I,J) of two-valued interpretations with $I \subseteq J$ is viewed as an *approximation* of some other two-valued interpretation: an atom interpreted as *true* (resp. *false*) by both I and J has a precise, exact value, namely *true* (resp. *false*); while an atom assigned to *false* by I but to *true* by J can be either *true* or *false*, i.e. it is *undefined*. In other words, a pair of two-valued interpretations is equivalent to a *three-valued interpretation* sending each atom to either *true*, *false*, or *undef*. Depending on the application, a suitable semantic operator, called an *approximator*, is defined on the set of three-valued interpretations, and then the machinery of AFT provides a family of well-known semantics and desired properties straightforwardly and clearly. It turns out that consistent AFT is suitable to define the stable model semantics for DatalogMTL$^\neg$, showcasing once again the power and versatility of the AFT formalism.

In this paper, we consider the set of pairs of interpretations already considered in [29], define an operator on it, and prove that such operator is a consistent approximator. Then, AFT allows us to obtain not only the *stable* model semantics for DatalogMTL$^\neg$, but also the *well-founded*, the *Kripke-Kleene*, and the *supported* model semantics as well. Finally, we prove that the stable models obtained via the usage of AFT are the same as the stable models defined in [29]. The advantages of employing the framework of AFT may be summarized as follows:

1. *Effortless.* The only thing required, is defining the right *approximator*; then the machinery of AFT lays out the stable, the well-founded, the Kripke-Kleene, and the supported model semantics for free. For most applications, the choice of an approximator is very natural; for DatalogMTL$^\neg$, it is no exception. The approximator, in fact, closely resembles Fitting's three-valued immediate consequence operator [14], with some slight adjustments to account for time points and datasets.
2. *Solid mathematical background.* AFT provides a broad range of properties and general results that hold for all its applications; hence, there is no need to re-prove several statements when switching settings. For instance, the minimality and uniqueness result (Theorem 3.4 in [29]) leading to the definition of stable models for DatalogMTL$^\neg$ in [29], is already guaranteed by the AFT framework.
3. *Reliable.* The strong mathematical foundations of AFT make the obtained results more reliable. Even when semantics have already been defined, as in the case of the stable model semantics for DatalogMTL$^\neg$, AFT constitutes a valuable tool to screen the body of previous work for errors. For example, the employment of AFT in the context of Abstract Dialectical Frameworks allowed the detection of some bugs in the original semantics [4].

The outline of the paper is as follows. First, we provide some preliminary material on both AFT (Section 2.1) and the syntax and semantics of DatalogMTL$^\neg$ (Section 2.2). In Section 3, we define the base notions we need to apply AFT: the three-valued semantics, and the immediate consequence operators (two- and three-valued). Moreover, we prove there exists a correspondence between the pre-fixpoints of these operators, and the two-valued and three-valued models. In Section 4, we are finally able to apply AFT; we just need to prove that the three-valued immediate consequence operator defined in the previous section is indeed a consistent approximator. Then, we are able to define the Kripke-Kleene,

stable, well-founded, and supported models via AFT. Finally, in Section 5, we compare our notions of three-valued models and two-valued stable models, with the concepts from [29], and we show that the stable models obtained via AFT are the same as the ones from [29].

2 Preliminaries

In this section, we first present the concepts that lays the foundations of AFT (Section 2.1). Then, we introduce the basic definitions and notation relative to timelines, the syntax of DatalogMTL$^\neg$, and the semantics of rule bodies (Section 2.2), as they were presented in [29].

2.1 AFT

AFT generalizes Tarki's theory to non-monotonic operators [27], with the initial goal of studying the semantics of non-monotonic logics. As such, AFT heavily relies on the following notions from order theory.

A *partially ordered set* (poset) P is a set equipped with a partial order $\leq_P$, i.e., a reflexive, antisymmetric, transitive relation. We often denote a poset by $\langle P, \leq_P \rangle$, where P is the underlying set, and $\leq_P$ the partial order. Given a subset $S \subseteq P$, a lower bound l of S is the *greatest lower bound of* S, denoted by $\bigsqcap S$, if it is greater than any other lower bound of S. Analogously, an upper bound u of S is the *least upper bound of* S, denoted by $\bigsqcup S$, if it is lower than any other upper bound of S. A *chain complete poset* (cpo) is a poset C such that for every chain $S \subseteq C$, i.e., a totally ordered subset, $\bigsqcup S$ exists. A *complete lattice* is a poset L such that for every subset $S \subseteq L$, both $\bigsqcup S$ and $\bigsqcap S$ exist. A function $f\colon P_1 \to P_2$ between posets is *monotone* if for all $x, y \in P_1$ such that $x \leq_{P_1} y$, it holds that $f(x) \leq_{P_2} f(y)$. We refer to functions $O\colon C \to C$ with domain equal to the codomain as *operators*. An element $x \in C$ is a *fixpoint* (resp. *pre-fixpoint*) of O if $O(x) = x$ (resp. $O(x) \leq_C x$). By Tarski's least fixpoint theorem, every monotone operator O on a cpo has a least fixpoint, denoted $\mathrm{lfp}(O)$. To use a similar principle for operators stemming from non-monotonic logics, consistent AFT [13] considers, for each complete lattice L, its associated *approximation space* $\langle L^c, \leq_p \rangle$, where $L^c = \{(x,y) \mid x, y \in L, x \leq_L y\} \subseteq L^2$, and $\leq_p$ is the *precision order* on the Cartesian product L^2, i.e., $(x_1, y_1) \leq_p (x_2, y_2)$ iff $x_1 \leq_L x_2$ and $y_2 \leq_L y_1$. L^c can be viewed as an approximation of L: an element $(x,y) \in L^c$ "approximates" all the values $z \in L$ such that $x \leq_L z \leq_L y$. Pairs of the form $(x,x) \in L^2$ are called *exact*, since they approximate only one element of L.

An operator $A\colon L^c \to L^c$ is a *consistent approximator of an operator* $O\colon L \to L$ if it is monotone and it behaves as O on exact pairs, i.e. $A(x,x) = (O(x), O(x))$ for all $x \in L$. We denote by A_1 and A_2 the first and second component of A, respectively, i.e. $A(x,y) = (A_1(x,y), A_2(x,y))$ for all $(x,y) \in L^c$. Since A is by definition monotone, by Tarski's theorem A has a least fixpoint, which is called the *Kripke-Kleene* fixpoint. Moreover, given an approximator A, it is easy to define three additional kinds of fixpoints of interests: *supported*, *stable*, and *well-founded* fixpoints. This will be done in detail at the end of Section 4, in Definition 4.1. If A is Fitting's three-valued immediate consequence operator [14] , then the aforementioned four types of fixpoint correspond to the homonymous semantics of logic programming [12, 11].

2.2 DatalogMTL$^\neg$

A *timeline* $\mathbb{T}$ is either the ring of integers $\langle \mathbb{Z}, + \rangle$, or the ring of rationals $\langle \mathbb{Q}, + \rangle$, with the order given by the addition. A *($\mathbb{T}$-)interval* δ is a non-empty subset of $\mathbb{T}$ such that the following conditions are satisfied:

1. δ is *convex*, i.e., for all $t_1, t_2 \in \delta$, for all $t \in \mathbb{T}$ such that $t_1 < t < t_2$, it holds that $t \in \delta$;

2. $\bigsqcap\delta, \bigsqcup\delta \in \mathbb{T} \cup \{+\infty, -\infty\}$.

We call an interval δ *non-negative* if for all $t \in \delta$, it holds that $t \geq 0$. We represent an interval δ with the standard notation $\langle \bigsqcap\delta, \bigsqcup\delta \rangle$, where the left bracket $\langle$ can either be [, if $\bigsqcap\delta \in \delta$, or (otherwise; and the right bracket $\rangle$ can either be], if $\bigsqcup\delta \in \delta$, or) otherwise. For any $t \in \mathbb{T}$, we denote the interval $[t,t]$ by simply t.

We now consider a function-free first-order vocabulary and a timeline $\mathbb{T}$. For any predicate P, and tuple $\mathbf{s}$ of constants and variables of the same arity of P, we call $P(\mathbf{s})$ a *relational atom*. A *metric atom* is an expression given by

$$M ::= \top \mid \bot \mid P(\mathbf{s}) \mid \diamondminus_\delta M \mid \boxminus_\delta M \mid \boxplus_\delta M \mid M \mathscr{S}_\delta M \mid M \mathscr{U}_\delta M,$$

where $P(\mathbf{s})$ ranges over relational atoms, and δ ranges over non-negative intervals. A metric atom is *ground* if it does not contain any variable. A *head atom* is a metric atom specified by the grammar:

$$M ::= \top \mid P(\mathbf{s}) \mid \boxminus_\delta M \mid \boxplus_\delta M, \tag{1}$$

where $P(\mathbf{s})$ ranges over relational atom, and δ ranges over non-negative intervals. We denote by $\mathscr{M}$ the set of ground head atoms. A *metric fact* is an expression $M@\delta$, where M is a ground metric atom, and δ is an interval. A metric fact $M@\delta$ is also a *relational fact* if M is a ground relational atom. A *dataset* is a finite set of relational facts. A *(two-valued) interpretation* I is a function which assigns to each $t \in \mathbb{T}$ a set $I(t)$ of ground relational atoms. A two-valued interpretation I is a *(two-valued) model of a dataset* $\mathscr{D}$ if for all relational facts $P(\mathbf{s})@\delta \in \mathscr{D}$, and for all $t \in \delta$, it holds that $P(\mathbf{s}) \in I(t)$. Notice that, interpretations can be ordered with the subset relation, i.e., $I \subseteq J$ if and only if for all $t \in \mathbb{T}$, $I(t) \subseteq J(t)$. We denote by $\mathbb{I}$ the set of all two-valued interpretations. A *rule* r is an expression of the form

$$M \leftarrow M_1 \wedge \ldots \wedge M_k \wedge \mathsf{not}\, M_{k+1} \wedge \ldots \wedge \mathsf{not}\, M_m, \tag{2}$$

where $m \geq k \geq 0$, each M_i is a metric atom, and M is a head atom. We call the atoms $M_1, \ldots, M_k$ *positive*, and $M_{k+1}, \ldots, M_m$ *negated*. The conjunction of positive and negated atoms of a rule is called its *body*, whereas the consequent M is its *head*. For brevity reasons, we will often denote a rule as $M \leftarrow B$, where B represents the conjunction in the body. Notice that rules of the form $\bot \leftarrow B$ are allowed in [29], but not in this paper. We will instead use rules of the form $P \leftarrow B \wedge \mathsf{not}\, P$ to express that B must be false in every model. A rule is *safe* if each variable it mentions in the head occurs in some positive body atom in a position other than a left operand of $\mathscr{S}$ or $\mathscr{U}$. A *program* is a finite set of safe rules. A rule is *ground* if it has no variables. For a program Π, we denote by $\mathsf{ground}(\Pi)$ the set of all the ground rules that can possibly be obtained from rules in Π by replacing variables with constants.

We conclude the preliminaries with the definitions of the *two-valued semantics* of DatalogMTL$^\neg$, and the *two valued models* of a dataset and a program.

Definition 2.1. *Let* Π *be a program, I an interpretation, and* $t \in \mathbb{T}$. *The* two-valued semantics *of ground metric atoms and bodies is defined as follows:*

1. $[\![\top]\!](I,t) = \mathit{true}$,
2. $[\![\bot]\!](I,t) = \mathit{false}$,
3. $[\![P(\mathbf{s})]\!](I,t) = \begin{cases} \mathit{true}, & \mathit{if}\ P(\mathbf{s}) \in I(t) \\ \mathit{false}, & \mathit{otherwise} \end{cases}$,
4. $[\![\diamondminus_\delta M]\!](I,t) = \bigsqcup_{\leq_\tau} \{[\![M]\!](I,t') \mid t' \in \mathbb{T}\ \mathit{such\ that}\ t - t' \in \delta\}$,

5. $[\![\oplus_\delta M]\!](I,t) = \bigsqcup_{\leq_\tau}\{[\![M]\!](I,t') \mid t' \in \mathbb{T} \text{ such that } t'-t \in \delta\}$,
6. $[\![\boxminus_\delta M]\!](I,t) = \bigsqcap_{\leq_\tau}\{[\![M]\!](I,t') \mid t' \in \mathbb{T} \text{ such that } t-t' \in \delta\}$,
7. $[\![\boxplus_\delta M]\!](I,t) = \bigsqcap_{\leq_\tau}\{[\![M]\!](I,t') \mid t' \in \mathbb{T} \text{ such that } t'-t \in \delta\}$,
8. $$[\![M_1 \mathscr{S}_\delta M_2]\!](I,t) = \begin{cases} true, & \text{if } [\![M_2]\!](I,t') = true \text{ for some } t' \text{ with } t-t' \in \delta \\ & \text{and } [\![M_1]\!](I,t'') = true \text{ for all } t'' \in (t',t) \\ false, & \text{otherwise} \end{cases},$$
9. $$[\![M_1 \mathscr{U}_\delta M_2]\!](I,t) = \begin{cases} true, & \text{if } [\![M_2]\!](I,t') = true \text{ for some } t' \text{ with } t'-t \in \delta \\ & \text{and } [\![M_1]\!](I,t'') = true \text{ for all } t'' \in (t,t') \\ false, & \text{otherwise} \end{cases},$$
10. $[\![\mathsf{not}\, M]\!](I,t) = [\![M]\!](I,t)^{-1}$,
11. $[\![M_1 \wedge \ldots \wedge M_m]\!](I,t) = \bigsqcap_{\leq_\tau}\{[\![M_1]\!](I,t), \ldots, [\![M_m]\!](I,t)\}$,

where we define $false^{-1} = true$ *and* $true^{-1} = false$*, and* $\leq_\tau$ *is the* truth order*, i.e. the order defined by* $false \leq_\tau true$*.*

Definition 2.2. *Let* Π *be a program,* $\mathscr{D}$ *a dataset, and* $I \in \mathbb{I}$ *an interpretation. Then, I is a* (two-valued) model *of* Π *iff for every rule* $M \leftarrow B \in \mathsf{ground}(\Pi)$*, and for every* $t \in \mathbb{T}$*,* $[\![B]\!](I,t) \leq_\tau [\![M]\!](I,t)$*. I is a* (two-valued) model *of* Π *and* $\mathscr{D}$ *iff I is a two-valued model of* Π *and a model of* $\mathscr{D}$*.*

3 The Three-valued Semantics of DatalogMTL$^\neg$ and the Immediate Consequence Operators

In order to apply the machinery of AFT, we just need to define an approximation space and a suitable approximator. This turns out to be fairly straightforward: the approximation space is the space of *three-valued interpretations*, and the approximator is an adaptation of Fitting's *three valued immediate consequence operator* [14].

In this section, we first define the three-valued interpretations and the three-valued semantics of rule bodies. Then, we define the two-valued, and the three-valued immediate consequence operators, and we show they characterize the two-valued, and three valued models, respectively.

Defining the three-valued semantics of rule bodies can easily be done by using three-valued interpretations. A *three-valued interpretation* $\mathscr{I}$ associates to each $t \in \mathbb{T}$ a function $\mathscr{I}(t)$ taking a ground relational atom and returning either *true*, *false*, or *undef*. Notice that three-valued interpretations may also be regarded as pairs of two-valued interpretations (I_1, I_2), where $I_1, I_2 \in \mathbb{I}$ such that $I_1 \subseteq I_2$: an atom in $I_1(t)$ is interpreted as *true*, an atom in $I_2(t)$ but not in $I_1(t)$ as *undef*, and an atom not in $I_2(t)$ as *false* (at time t). Notice that we have the following correspondence of truth values between the three-valued approach and the "pairs" approach: $true \mathrel{\widehat{=}} (true, true)$, $undef \mathrel{\widehat{=}} (false, true)$, and $false \mathrel{\widehat{=}} (false, false)$. For the sake of clarity, we will use caligraphic fonts to differentiate three-valued interpretations from two-valued ones. For a three-valued interpretation $\mathscr{I}$, we will use the notation I_1 and I_2 for the two two-valued interpretations forming the pair $\mathscr{I}$ corresponds to., i.e., $\mathscr{I} = (I_1, I_2)$. We will denote by $\mathbb{I}^c$ the set of three-valued interpretations, following the notation introduced in Section 2.1. We can now define the three-valued semantics straightforwardly. Notice that we employ the same notation used for the two-valued semantics in Definition 2.1; the two semantics are easy to distinguish by looking at the interpretation (two- or three-valued) in the argument.

Definition 3.1. *Let* Π *be a program,* $\mathscr{I}$ *a three-valued interpretation, and* $t \in \mathbb{T}$. *The* three-valued semantics *of ground metric atoms and bodies is defined as follows:*

1. $[\![M]\!](\mathscr{I},t) = ([\![M]\!](I_1,t), [\![M]\!](I_2,t))$,
2. $[\![\mathsf{not}\, M]\!](\mathscr{I},t) = [\![M]\!](\mathscr{I},t)^{-1}$,
3. $[\![E_1 \wedge \ldots \wedge E_m]\!](\mathscr{I},t) = \bigsqcap_{\leq_\tau}\{[\![E_1]\!](\mathscr{I},t), \ldots, [\![E_m]\!](\mathscr{I},t)\}$,

where we define $false^{-1} = true$, $undef^{-1} = undef$, *and* $true^{-1} = false$, *and* $\leq_\tau$ *is the (three-valued)* truth order, *i.e. the order defined by* $false \leq_\tau undef \leq_\tau true$.

Remark 1. *Notice that in Definition 3.1 it holds that*

$$[\![E_1 \wedge \ldots \wedge E_m]\!](\mathscr{I},t) = \bigsqcap_{\leq_\tau}\{[\![E_1]\!](\mathscr{I},t), \ldots, [\![E_m]\!](\mathscr{I},t)\} = ([\![E_1 \wedge \ldots \wedge E_m]\!](I_1,t), [\![E_1 \wedge \ldots \wedge E_m]\!](I_2,t)).$$

Analogously to what we did in Section 2.2, we naturally obtain the definition of *three-valued model*.

Definition 3.2. *Let* Π *be a program,* $\mathscr{D}$ *a dataset, and* $\mathscr{I} \in \mathbb{I}^{\mathsf{c}}$ *a three-valued interpretation. Then,* $\mathscr{I}$ *is a* (three-valued) model *of* Π *iff for every rule* $M \leftarrow B \in \mathsf{ground}(\Pi)$, *and for every* $t \in \mathbb{T}$, $[\![B]\!](\mathscr{I},t) \leq_\tau [\![M]\!](\mathscr{I},t)$. $\mathscr{I}$ *is a* (three-valued) model *of* Π *and* $\mathscr{D}$ *iff* $\mathscr{I}$ *is a three-valued model of* Π *and* I_1 *is a model of* $\mathscr{D}$.

Before defining the two-valued and the three-valued immediate consequence operators, we build a function F to make explicit the link between a ground head atom and the time points it refers to. Recall that we allow the nesting of $\boxplus$ and $\boxminus$ operators in the head of rules.

For all $t \in \mathbb{T}$, and for all ground head atoms $M \in \mathscr{M}$, we define:

$$F(M,t) := \begin{cases} \bigcup_{t' \in \mathbb{T} \mid t - t' \in \delta}\{F(M',t')\} & \text{if } M = \boxminus_\delta M' \\ \bigcup_{t' \in \mathbb{T} \mid t' - t \in \delta}\{F(M',t')\} & \text{if } M = \boxplus_\delta M' \\ \{(M,t)\} & \text{otherwise} \end{cases}.$$

From the definitions of the two-valued and the three-valued semantics, and the construction of F, it is easy to see the relation between a ground head atom M at time t and the relational facts it refers to, as expressed more precisely in the following propositions.

Proposition 3.1. *Let* $M \in \mathscr{M}$, $\mathscr{I} \in \mathbb{I}^{\mathsf{c}}$, *and* $t \in \mathbb{T}$. *The following statements hold:*

1. $[\![M]\!](I,t) = true$ *if and only if* $[\![P(\mathbf{s})]\!](I,t') = true$ *for all* $(P(\mathbf{s}),t') \in F(M,t)$.
2. $[\![M]\!](\mathscr{I},t) = \bigsqcap_{\leq_\tau}\{[\![P(\mathbf{s})]\!](\mathscr{I},t') \mid (P(\mathbf{s}),t') \in F(M,t)\}$.

Proof. Item 1 is obvious by the definition of F. For Item 2, by Definition 3.1, we can write the following equalities: $[\![M]\!](\mathscr{I},t) = ([\![M]\!](I_1,t), [\![M]\!](I_2,t))$, and $[\![P(\mathbf{s})]\!](\mathscr{I},t') = ([\![P(\mathbf{s})]\!](I_1,t'), [\![P(\mathbf{s})]\!](I_2,t'))$ for all $(P(\mathbf{s}),t') \in F(M,t)$. Then Item 2 follows easily from Item 1. □

Now, the definition of the two-valued, and three-valued immediate consequence operators for a database $\mathscr{D}$ and a program Π naturally follow.

Definition 3.3. *Let* Π *be a program, and* $\mathscr{D}$ *be a dataset. The mapping* $T_{\mathscr{D},\Pi} \colon \mathbb{I} \to \mathbb{I}$ *is called the* immediate consequence operator for $\mathscr{D}$ and Π, *and it is defined for all* $t \in \mathbb{T}$ *as follows*

$$\begin{aligned} T_{\mathscr{D},\Pi}(I)(t) :=& \{P(\mathbf{s}) \mid \exists \delta \text{ such that } P(\mathbf{s})@\delta \in \mathscr{D} \text{ and } t \in \delta\} \\ &\cup \{P(\mathbf{s}) \mid \exists M \leftarrow B \in \mathsf{ground}(\Pi), \exists t' \in \mathbb{T} \text{ such that } [\![B]\!](I,t') = true \text{ and } (P(\mathbf{s}),t) \in F(M,t')\} \end{aligned}$$

Definition 3.4. *Let Π be a program, and $\mathscr{D}$ be a dataset. The mapping $\Psi_{\mathscr{D},\Pi}\colon \mathbb{I}^{\mathsf{c}} \to \mathbb{I}^{\mathsf{c}}$ is called the* three-valued immediate consequence operator for $\mathscr{D}$ and Π*, and its components $\Psi^1_{\mathscr{D},\Pi}\colon \mathbb{I}^{\mathsf{c}} \to \mathbb{I}$, and $\Psi^2_{\mathscr{D},\Pi}\colon \mathbb{I}^{\mathsf{c}} \to \mathbb{I}$ are defined for all $t \in \mathbb{T}$ as follows*

$$\begin{aligned}\Psi^1_{\mathscr{D},\Pi}(\mathscr{I})(t) :=& \{P(\mathbf{s}) \mid \exists \delta \textit{ such that } P(\mathbf{s})@\delta \in \mathscr{D} \textit{ and } t \in \delta\}\\ &\cup \{P(\mathbf{s}) \mid \exists M \leftarrow B \in \mathsf{ground}(\Pi), \exists t' \in \mathbb{T} \textit{ such that } [\![B]\!](\mathscr{I},t') = \textit{true} \textit{ and } (P(\mathbf{s}),t) \in F(M,t')\},\\ \Psi^2_{\mathscr{D},\Pi}(\mathscr{I})(t) :=& \{P(\mathbf{s}) \mid \exists \delta \textit{ such that } P(\mathbf{s})@\delta \in \mathscr{D} \textit{ and } t \in \delta\}\\ &\cup \{P(\mathbf{s}) \mid \exists M \leftarrow B \in \mathsf{ground}(\Pi), \exists t' \in \mathbb{T} \textit{ such that } [\![B]\!](\mathscr{I},t') \neq \textit{false} \textit{ and } (P(\mathbf{s}),t) \in F(M,t')\},\end{aligned}$$

Finally, as expected, $T_{\mathscr{D},\Pi}$ and $\Psi_{\mathscr{D},\Pi}$ characterize the two-valued, and the three-valued models, respectively.

Proposition 3.2. *Let Π be a program, $\mathscr{D}$ a database, and $I \in \mathbb{I}$. The interpretation I is a two-valued model of $\mathscr{D}$ and Π if and only if I is a pre-fixpoint of $T_{\mathscr{D},\Pi}$.*

Proof. Suppose that I is a two-valued model of $\mathscr{D}$ and Π. We have to show that for all $t \in \mathbb{T}$, $T_{\mathscr{D},\Pi}(I)(t) \subseteq I(t)$. Let $P(\mathbf{s}) \in T_{\mathscr{D},\Pi}(I)(t)$. If there exists δ such that $t \in \delta$ and $P(\mathbf{s})@\delta \in \mathscr{D}$, then $P(\mathbf{s}) \in I(t)$ because I is, in particular, a model of $\mathscr{D}$. Let us now assume that such an interval δ does not exist. Then, it must be the case that there exists a ground rule $M \leftarrow B \in \mathsf{ground}(\Pi)$ and a time point $t' \in \mathbb{T}$ such that $[\![B]\!](I,t') = \textit{true}$ and $(P(\mathbf{s}),t) \in F(M,t')$. Since I is a two-valued model of Π, it holds that $[\![B]\!](I,t') \leq_\tau [\![M]\!](I,t')$; hence, it must hold that $[\![M]\!](I,t') = \textit{true}$. By Proposition 3.1, it follows that $[\![P(\mathbf{s})]\!](I,t) = \textit{true}$, i.e., $P(\mathbf{s}) \in I(t)$, as desired.

Conversely, assume that for all $t \in \mathbb{T}$, $T_{\mathscr{D},\Pi}(I)(t) \subseteq I(t)$. By definition of $T_{\mathscr{D},\Pi}$, for all ground relational facts $P(\mathbf{s})@\delta \in \mathscr{D}$, and for all $t \in \delta$, it holds that $P(\mathbf{s}) \in T_{\mathscr{D},\Pi}(I)(t) \subseteq I(t)$. Hence, I is a model of $\mathscr{D}$. We now prove that I is also a two-valued model of Π. Let $M \leftarrow B \in \mathsf{ground}(\Pi)$, and $t \in \mathbb{T}$, and suppose $[\![B]\!](I,t) = \textit{true}$. It suffices to show that $[\![M]\!](I,t) = \textit{true}$. If $M = \top$, this is trivially verified. Suppose now that $M \neq \top$. Observe that, by definition of $T_{\mathscr{D},\Pi}$, for all $(P(\mathbf{s}),t') \in F(M,t)$, $P(\mathbf{s}) \in T_{\mathscr{D},\Pi}(I)(t') \subseteq I(t')$. In particular, $[\![P(\mathbf{s})]\!](I,t') = \textit{true}$ for all $(P(\mathbf{s}),t') \in F(M,t)$. Hence, by Proposition 3.1, $[\![M]\!](I,t) = \textit{true}$, as desired. □

Proposition 3.3. *Let Π be a program, $\mathscr{D}$ a database, and $\mathscr{I} \in \mathbb{I}^{\mathsf{c}}$. The interpretation $\mathscr{I}$ is a three-valued model of $\mathscr{D}$ and Π if and only if $\mathscr{I}$ is a pre-fixpoint of $\Psi_{\mathscr{D},\Pi}$.*

Proof. Suppose that $\mathscr{I}$ is a three-valued model of $\mathscr{D}$ and Π. We have to show that $\Psi_{\mathscr{D},\Pi}(\mathscr{I}) \leq_\tau \mathscr{I}$. Let $P(\mathbf{s})$ be a ground relational atom such that $\Psi_{\mathscr{D},\Pi}(\mathscr{I})(t)(P(\mathbf{s})) = \textit{true}$. Hence, by Definition 3.4, either there exists an interval δ such that $t \in \delta$ and $P(\mathbf{s})@\delta \in \mathscr{D}$, in which case $\mathscr{I}(P(\mathbf{s})) = \textit{true}$ (as $\mathscr{I}$ is a model of $\mathscr{D}$ by hypothesis); or there exists a ground rule $M \leftarrow B \in \mathsf{ground}(\Pi)$ and a time point $t' \in \mathbb{T}$ such that $[\![B]\!](\mathscr{I},t') = \textit{true}$ and $(P(\mathbf{s}),t) \in F(M,t')$. Since $\mathscr{I}$ is a three-valued model of Π, it holds that $[\![B]\!](\mathscr{I},t') \leq_\tau [\![M]\!](\mathscr{I},t')$; hence, it must hold that $[\![M]\!](\mathscr{I},t') = \textit{true}$. By Proposition 3.1, it follows that $\mathscr{I}(t)(P(\mathbf{s})) = (I^1(t)(P(\mathbf{s})), I^2(t)(P(\mathbf{s})) = ([\![P(\mathbf{s})]\!](I^1,t), [\![P(\mathbf{s})]\!](I^2,t)) = [\![P(\mathbf{s})]\!](\mathscr{I},t) = \textit{true}$, as desired. Now let $P(\mathbf{s})$ be a ground relational atom such that $\Psi_{\mathscr{D},\Pi}(\mathscr{I})(t)(P(\mathbf{s})) = \textit{undef}$. Hence, by Definition 3.4, there exists a ground rule $M \leftarrow B \in \mathsf{ground}(\Pi)$ and a time point $t' \in \mathbb{T}$ such that $[\![B]\!](\mathscr{I},t') \neq \textit{false}$ and $(P(\mathbf{s}),t) \in F(M,t')$. Since $\mathscr{I}$ is a three-valued model of Π, it holds that $\textit{undef} \leq_\tau [\![B]\!](\mathscr{I},t') \leq_\tau [\![M]\!](\mathscr{I},t')$. In particular, we have that $[\![M]\!](\mathscr{I},t') \neq \textit{false}$. By Proposition 3.1, it follows that $\mathscr{I}(t)(P(\mathbf{s})) = [\![P(\mathbf{s})]\!](\mathscr{I},t) = \textit{false}$, concluding the first half of the proof.

Conversely, assume that for all $t \in \mathbb{T}$, $\Psi_{\mathscr{D},\Pi}(\mathscr{I})(t) \leq_\tau \mathscr{I}(t)$. By definition of $\Psi_{\mathscr{D},\Pi}$, for all ground relational facts $P(\mathbf{s})@\delta \in \mathscr{D}$, and for all $t \in \delta$, it holds that $\textit{true} = \Psi_{\mathscr{D},\Pi}(\mathscr{I})(t)(P(\mathbf{s})) \leq_\tau \mathscr{I}(t)(P(\mathbf{s}))$.

Hence, $\mathscr{I}$ is a model of $\mathscr{D}$. We now prove that $\mathscr{I}$ is also a three-valued model of Π. Let $M \leftarrow B \in \mathsf{ground}(\Pi)$, and $t \in \mathbb{T}$. We have to show that $[\![B]\!](\mathscr{I},t) \leq_\tau [\![M]\!](\mathscr{I},t)$. If $M = \top$, this is trivially verified, so assume that $M \neq \top$. We first suppose $[\![B]\!](\mathscr{I},t) = \mathit{true}$ and show that $[\![M]\!](\mathscr{I},t) = \mathit{true}$. Observe that, by definition of $\Psi_{\mathscr{D},\Pi}$, for all $(Q(\mathbf{s}),t') \in F(M,t)$, it holds that $\mathit{true} = \Psi_{\mathscr{D},\Pi}(\mathscr{I})(t')(Q(\mathbf{s})) \leq_\tau \mathscr{I}(t')(Q(\mathbf{s}))$. In particular, $[\![Q(\mathbf{s})]\!](\mathscr{I},t') = \mathit{true}$ for all $(Q(\mathbf{s}),t') \in F(M,t)$. Hence, by Proposition 3.1, $[\![M]\!](\mathscr{I},t) = \mathit{true}$, as desired. Now, we assume $[\![B]\!](\mathscr{I},t) = \mathit{undef}$ and show that $[\![M]\!](\mathscr{I},t) \neq \mathit{false}$. Observe that, by definition of $\Psi_{\mathscr{D},\Pi}$, for all $(Q(\mathbf{s}),t') \in F(M,t)$, it holds that $\mathit{undef} \leq_\tau \Psi_{\mathscr{D},\Pi}(\mathscr{I})(t')(Q(\mathbf{s})) \leq_\tau \mathscr{I}(t')(Q(\mathbf{s}))$. In particular, $[\![Q(\mathbf{s})]\!](\mathscr{I},t') \geq_\tau \mathit{undef}$ for all $(Q(\mathbf{s}),t') \in F(M,t)$. Hence, by Proposition 3.1, $[\![M]\!](\mathscr{I},t) \geq_\tau \mathit{undef}$, concluding the proof. □

4 Approximation Fixpoint Theory for DatalogMTL$^\neg$

After having defined the three-valued semantics and the three-valued immediate consequence operator in the previous section, it only remains to show that the three-valued immediate consequence operator $\Psi_{\mathscr{D},\Pi}$ is a consistent approximator of $T_{\mathscr{D},\Pi}$ (Theorem 4.1). Then, the machinery of AFT allows for the definition of four kinds of models of a database $\mathscr{D}$ and a program Π: *supported*, *stable*, *Kripke-Kleene*, and *well-founded*.

We start by proving that the space of two-valued interpretations $\mathbb{I}$ is a complete lattice, which follows straightforwardly from results of order theory.

Proposition 4.1. *$\langle \mathbb{I}, \subseteq \rangle$ is a complete lattice.*

Proof. Let $\mathscr{G}$ be the set of ground relational atoms. Then $\mathbb{I}$ is the set of functions with signature $\mathbb{T} \to 2^{\mathscr{G}}$. The power set $2^{\mathscr{G}}$ ordered with set inclusion $\subseteq$ is clearly a complete lattice, with bottom element the empty set, and top element $\mathscr{G}$. Moreover, $\mathbb{T}$ can be considered as a poset with the standard order on integers or rationals. Since the category of complete lattices is a full subcategory of the category of posets, by Proposition 2 in [25], we have the isomorphism $\mathbb{I} \cong \Pi_{t\in\mathbb{T}} 2^{\mathscr{G}}$. By Proposition 1 in [25], $\Pi_{t\in\mathbb{T}} 2^{\mathscr{G}}$ is a complete lattice, hence, $\mathbb{I}$ is a complete lattice too. □

Using the terminology of AFT, the set of three-valued interpretations $\mathbb{I}^c$ equipped with the precision order $\leq_p$ ($\mathscr{I} \leq_p \mathscr{J}$ iff $I_1 \subseteq J_1 \subseteq J_2 \subseteq I_2$) is an *approximation space* for $\mathbb{I}$. In order to prove Theorem 4.1, we make use of the following two lemmata, which ensure the evaluation of ground metric atoms preserves the order relation in both the two-valued and the three-valued case.

Lemma 4.1. *Let $I_1, I_2 \in \mathbb{I}$ be two-valued interpretations, and M be a ground atom. If $I_1 \subseteq I_2$, then for all $t \in \mathbb{T}$, $[\![M]\!](I_1,t) \leq_\tau [\![M]\!](I_2,t)$.*

Proof. Let $I_1 \subseteq I_2$, i.e., $I_1(t) \subseteq I_2(t)$ for all $t \in \mathbb{T}$. We proceed by induction on the structure of M.

If $M \in \{\bot, \top\}$, then clearly $[\![M]\!](I_1,t) = [\![M]\!](I_2,t)$. If $M = P(\mathbf{s})$, then $[\![M]\!](I_1,t) \leq_\tau [\![M]\!](I_2,t)$ follows from Definition 2.1 and the assumption that $I_1(t) \subseteq I_2(t)$.

Now, let $M = \diamondsuit_\delta M'$, and assume $[\![M']\!](I_1,t) \leq_\tau [\![M']\!](I_2,t)$ for all $t \in \mathbb{T}$. If $[\![M]\!](I_1,t) = \mathit{true}$ for some $t \in \mathbb{T}$, then by Definition 2.1, there exists $t' \in \mathbb{T}$ such that $t - t' \in \delta$ for which $[\![M']\!](I_1,t') = \mathit{true}$. By induction hypothesis, it must hold that $[\![M']\!](I_2,t') = \mathit{true}$, as desired. We can proceed analogously for the cases $M = \blacklozenge_\delta M'$, $M = \boxminus_\delta M'$, and $M = \boxplus_\delta M'$.

Let $M = M_1 \mathscr{S}_\delta M_2$, and assume $[\![M_1]\!](I_1,t) \leq_\tau [\![M_1]\!](I_2,t)$, and $[\![M_2]\!](I_1,t) \leq_\tau [\![M_2]\!](I_2,t)$ for all $t \in \mathbb{T}$. If $[\![M]\!](I_1,J_1,t) = \mathit{true}$ for some $t \in \mathbb{T}$, then by Definition 2.1, there exists $t' \in \mathbb{T}$ such that $t - t' \in \delta$, $[\![M_2]\!](I_1,t') = \mathit{true}$, and $[\![M_1]\!](I_1,t'') = \mathit{true}$ for all $t'' \in (t',t)$. By induction hypothesis, it must hold that

$[\![M_2]\!](I_2,t') = \mathit{true}$, and $[\![M_1]\!](I_2,t'') = \mathit{true}$ for all $t'' \in (t',t)$. It follows that $[\![M]\!](I_2,t) = \mathit{true}$, as desired. We can proceed analogously for the case $M = M_1 \mathscr{U}_\delta M_2$. □

Lemma 4.2. *Let $\mathscr{I}, \mathscr{J} \in \mathbb{I}^c$ be three-valued interpretations, and E be a ground expression. If $\mathscr{I} \leq_p \mathscr{J}$, then for all $t \in \mathbb{T}$, $[\![E]\!](\mathscr{I},t) \leq_p [\![E]\!](\mathscr{J},t)$.*

Proof. Assume that $\mathscr{I} \leq_p \mathscr{J}$, i.e., $I_1(t) \subseteq J_1(t) \subseteq J_2(t) \subseteq I_2(t)$ for all $t \in \mathbb{T}$. We proceed by induction on the structure of E.

If E is a ground metric atom, then by Lemma 4.1, we have $[\![E]\!](I_1,t) \leq_\tau [\![E]\!](J_1,t) \leq_\tau [\![E]\!](J_2,t) \leq_\tau [\![E]\!](I_2,t)$. By Definition 3.1, we obtain $[\![E]\!](\mathscr{I},t) \leq_p [\![E]\!](\mathscr{J},t)$.

Now assume $E = \mathsf{not}\ M$ for some ground metric atom M, and that $[\![M]\!](\mathscr{I},t) \leq_p [\![M]\!](\mathscr{J},t)$. If $[\![M]\!](\mathscr{I},t) = [\![M]\!](\mathscr{J},t)$, then clearly $[\![E]\!](\mathscr{I},t) = [\![E]\!](\mathscr{J},t)$. Now assume $[\![M]\!](\mathscr{I},t) <_p [\![M]\!](\mathscr{J},t)$. It must be that $[\![M]\!](\mathscr{I},t) = \mathit{undef}$ and $[\![M]\!](\mathscr{J},t) \in \{\mathit{true},\mathit{false}\}$. By Definition 3.1, it clearly follows that $[\![E]\!](\mathscr{I},t) = \mathit{undef}$ and $[\![E]\!](\mathscr{J},t) \in \{\mathit{true},\mathit{false}\}$; hence $[\![E]\!](\mathscr{I},t) \leq_p [\![E]\!](\mathscr{J},t)$.

Finally, for the case $E = E_1 \wedge \ldots \wedge E_m$, we can easily use the induction hypothesis on $E_1,\ldots,E_m$ to obtain that $[\![E]\!](\mathscr{I},t) \leq_p [\![E]\!](\mathscr{J},t)$. □

Theorem 4.1. *Let Π be a program, and $\mathscr{D}$ be a dataset. The operator $\Psi_{\mathscr{D},\Pi}$ is a consistent approximator of $T_{\mathscr{D},\Pi}$.*

Proof. We have to prove three things:

1. $\Psi_{\mathscr{D},\Pi}$ *is consistent*, i.e., for all $\mathscr{I} \in \mathbb{I}^c$, $\Psi_{\mathscr{D},\Pi}(\mathscr{I}) \in \mathbb{I}^c$.
2. $\Psi_{\mathscr{D},\Pi}$ *is $\leq_p$-monotone,* i.e., for all $\mathscr{I}, \mathscr{J} \in \mathbb{I}^c$ such that $\mathscr{I} \leq_p \mathscr{J}$, it holds that $\Psi_{\mathscr{D},\Pi}(\mathscr{I}) \leq_p \Psi_{\mathscr{D},\Pi}(\mathscr{J})$.
3. $\Psi_{\mathscr{D},\Pi}$ *approximates $T_{\mathscr{D},\Pi}$,* i.e., for all $I \in \mathbb{I}$, $\Psi_{\mathscr{D},\Pi}(I,I) = (T_{\mathscr{D},\Pi}(I), T_{\mathscr{D},\Pi}(I))$.

For Item 1, observe that from Definition 3.4 we clearly have $\Psi^1_{\mathscr{D},\Pi}(\mathscr{I})(t) \subseteq \Psi^2_{\mathscr{D},\Pi}(\mathscr{I})(t)$ for all $\mathscr{I} \in \mathbb{I}^c$ and for all $t \in \mathbb{T}$. Hence, for all $\mathscr{I} \in \mathbb{I}^c$, it holds that $\Psi_{\mathscr{D},\Pi}(\mathscr{I}) \in \mathbb{I}^c$, as desired.

Now, to prove Item 2, let $\mathscr{I} \leq_p \mathscr{J}$ be two elements of $\mathbb{I}^c$, and $t \in \mathbb{T}$. We need to show that $\Psi^1_{\mathscr{D},\Pi}(\mathscr{I})(t) \subseteq \Psi^1_{\mathscr{D},\Pi}(\mathscr{J})(t)$ and $\Psi^2_{\mathscr{D},\Pi}(\mathscr{J})(t) \subseteq \Psi^2_{\mathscr{D},\Pi}(\mathscr{I})(t)$. Let $P(\mathbf{s}) \in \Psi^1_{\mathscr{D},\Pi}(\mathscr{I})(t)$ (if there is no such $P(\mathbf{s})$, then we already have $\Psi^1_{\mathscr{D},\Pi}(\mathscr{I})(t) \subseteq \Psi^1_{\mathscr{D},\Pi}(\mathscr{J})(t)$). If there exists δ such that $t \in \delta$ and $P(\mathbf{s})@\delta \in \mathscr{D}$, then $P(\mathbf{s}) \in \Psi^1_{\mathscr{D},\Pi}(\mathscr{J})(t)$ too. If this is not the case, then there exists a ground rule $M \leftarrow B \in \mathsf{ground}(\Pi)$ and $t' \in \mathbb{T}$ such that $[\![B]\!](\mathscr{I},t') = \mathit{true}$ and $(P(\mathbf{s}),t) \in F(M,t')$. By Lemma 4.2, it holds that $[\![B]\!](\mathscr{J},t') = \mathit{true}$, hence $P(\mathbf{s}) \in \Psi^1_{\mathscr{D},\Pi}(\mathscr{J})(t)$. We have so far proved the first inclusion, i.e., $\Psi^1_{\mathscr{D},\Pi}(\mathscr{I})(t) \subseteq \Psi^1_{\mathscr{D},\Pi}(\mathscr{J})(t)$. Showing that the second inclusion holds can be performed analogously.

Finally, for Item 3, let B be the body of a ground rule in $\mathsf{ground}(\Pi)$, and $t \in \mathbb{T}$. It is easy to prove by induction on B, and using Definition 3.1 and Remark 1, that $[\![B]\!]((I,I),t) = ([\![B]\!](I,t), [\![B]\!](I,t))$. In particular, $[\![B]\!]((I,I),t) \neq \mathit{undef}$, and $[\![B]\!]((I,I),t) = \mathit{true}$ if and only if $[\![B]\!]((I,I),t) \neq \mathit{false}$. Hence, by Definitions 3.3 and 3.4, for all $t \in \mathbb{T}$, $\Psi^1_{\mathscr{D},\Pi}(I,I)(t) = \Psi^2_{\mathscr{D},\Pi}(I,I)(t) = T_{\mathscr{D},\Pi}(I)(t)$, as desired. □

Thanks to Theorem 4.1, and the machinery of AFT, we are able to define the following models of interest.

Definition 4.1. *Let $\mathscr{D}$ be a dataset, Π be a program, and $\mathscr{I} \in \mathbb{I}^c$.*

- *$\mathscr{I}$ is a* three-valued supported model of $\mathscr{D}$ and Π *if it is a fixpoint of $\Psi_{\mathscr{D},\Pi}$;*

- $\mathscr{I}$ *is a* three-valued stable model of $\mathscr{D}$ and Π *if it is a stable fixpoint of* $\Psi_{\mathscr{D},\Pi}$*; that is, if* $I_1 = \operatorname{lfp}\Psi^1_{\mathscr{D},\Pi}(\cdot, I_2)$ *and* $I_2 = \operatorname{lfp}\Psi^2_{\mathscr{D},\Pi}(I_1,\cdot)$*, where* $\Psi^1_{\mathscr{D},\Pi}(\cdot,I_2)$ *is the function that maps an interpretation* X *to the first component of* $\Psi_{\mathscr{D},\Pi}(X,J)$*, and similarly for* $\Psi^2_{\mathscr{D},\Pi}(I_1,\cdot)$*;*
- $\mathscr{I}$ *is the* Kripke-Kleene model of $\mathscr{D}$ and Π *if it is the* $\leq_p$*-least fixpoint of* $\Psi_{\mathscr{D},\Pi}$*;*
- $\mathscr{I}$ *is the* well-founded model of $\mathscr{D}$ and Π *if it is the well-founded fixpoint of* $\Psi_{\mathscr{D},\Pi}$*, i.e., if it is the* $\leq_p$*-least three-valued stable model.*

We call a two-valued model I *of* $\mathscr{D}$ *and* Π *a* two-valued stable model of $\mathscr{D}$ and Π *if* (I,I) *is a three-valued stable model of* $\mathscr{D}$ *and* Π*.*

5 Comparison with Previous Work

In this section, we compare the definitions of *HT-models* and stable models of [29] with ours. In particular, we first show that any three-valued model is an HT-model from [29], while the converse does not hold in general. Finally, we prove that indeed the notion of *stable model* from [29] coincides with our definition of two-valued stable model. For the sake of clarity, we rewrite the definitions of HT-model (Definition 5.1) and stable model (Definition 5.2) from [29] with the notation used in this paper. To avoid ambiguity, we will rename the stable models from [29] "stable HT-models".

Definition 5.1. *Let* $\mathscr{D}$ *be a dataset,* Π *be a program, and* $\mathscr{I} \in \mathbb{I}^{\mathsf{c}}$*.* $\mathscr{I}$ *is an* HT-model of $\mathscr{D}$ and Π *if* I_1 *is a model of* $\mathscr{D}$ *and for all rules* $M \leftarrow M_1 \wedge \ldots \wedge M_k \wedge \mathsf{not}\, M_{k+1} \wedge \ldots \wedge \mathsf{not}\, M_m \in \mathsf{ground}(\Pi)$*, for all* $t \in \mathbb{T}$*, the following conditions hold*

1. *if* $[\![M_i]\!](I_1,t) = \mathit{true}$ *for all* $i \in \{1,\ldots,k\}$ *and* $[\![M_j]\!](I_2,t) = \mathit{false}$ *for all* $j \in \{k+1,\ldots,m\}$*, then* $[\![M]\!](I_1,t) = \mathit{true}$*;*
2. *if* $[\![M_i]\!](I_2,t) = \mathit{true}$ *for all* $i \in \{1,\ldots,k\}$ *and* $[\![M_j]\!](I_2,t) = \mathit{false}$ *for all* $j \in \{k+1,\ldots,m\}$*, then* $[\![M]\!](I_2,t) = \mathit{true}$*.*

Proposition 5.1. *Let* $\mathscr{D}$ *be a dataset,* Π *be a program, and* $\mathscr{I} \in \mathbb{I}^{\mathsf{c}}$*. If* $\mathscr{I}$ *is a three-valued model of* $\mathscr{D}$ *and* Π*, then* $\mathscr{I}$ *is an HT-model of* $\mathscr{D}$ *and* Π*.*

Proof. Clearly, I_1 is a model for $\mathscr{D}$ by hypothesis. We now show that Item 1 from Definition 5.1 holds. Let $M \leftarrow B \in \mathsf{ground}(\Pi)$ be a ground rule with $B = M_1 \wedge \ldots \wedge M_k \wedge \mathsf{not}\, M_{k+1} \wedge \ldots \wedge \mathsf{not}\, M_m$, and $t \in \mathbb{T}$ such that $[\![M_i]\!](I_1,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](I_2,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. This implies that $[\![M_i]\!](\mathscr{I},t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](\mathscr{I},t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. In particular, by Defintion 3.1 we have that $[\![B]\!](\mathscr{I},t) = \mathit{true}$. Since $\mathscr{I}$ is a three-valued model by hypothesis, it must be that $[\![M]\!](\mathscr{I},t) = \mathit{true}$ too. Hence, $[\![M]\!](I_1,t) = \mathit{true}$, as desired.

Now, we prove that Item 2 from Definition 5.1 holds as well. Let $M \leftarrow B \in \mathsf{ground}(\Pi)$ be a ground rule with $B = M_1 \wedge \ldots \wedge M_k \wedge \mathsf{not}\, M_{k+1} \wedge \ldots \wedge \mathsf{not}\, M_m$, and $t \in \mathbb{T}$ such that $[\![M_i]\!](I_2,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](I_2,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. This implies that $[\![M_i]\!](\mathscr{I},t) \neq \mathit{false}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](\mathscr{I},t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. In particular, by Definition 3.1 we have that $[\![B]\!](\mathscr{I},t) \neq \mathit{false}$. Since $\mathscr{I}$ is a three-valued model, it must be that $[\![M]\!](\mathscr{I},t) \neq \mathit{false}$. Hence, $[\![M]\!](I_2,t) = \mathit{true}$, concluding the proof. □

The converse of Proposition 5.1 does not hold, i.e. not all HT-models are three-valued models. In fact, by Definition 3.2, $\mathscr{I} \in \mathbb{I}^{\mathsf{c}}$ is a three-valued model of Π iff for every rule $M \leftarrow B \in \mathsf{ground}(\Pi)$, and for every $t \in \mathbb{T}$, it holds that $[\![B]\!](\mathscr{I},t) \leq_\tau [\![M]\!](\mathscr{I},t)$, i.e. it holds that

1. if $[\![M_i]\!](I_1,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](I_2,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$, then $[\![M]\!](I_1,t) = \mathit{true}$,
2. if $[\![M_i]\!](I_2,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](I_1,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$, then $[\![M]\!](I_2,t) = \mathit{true}$,

where $B := M_1 \wedge \ldots \wedge M_k \wedge \mathsf{not}\ M_{k+1} \wedge \ldots \wedge \mathsf{not}\ M_m$. Notice that, while the first condition above is the same as the first one in Definition 5.1, the second condition is more strict than the second condition of Definition 5.1: it applies even when $[\![M_h]\!](I_2,t) = \mathit{true}$ for some $h \in \{k+1,\ldots,m\}$, provided that $[\![M_j]\!](I_1,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$.

We show this discrepancy more concretely in the example below.

Example 5.1. *Consider an empty dataset $\mathscr{D}$ and a program Π with just one rule, namely $P \leftarrow \mathsf{not}\ Q$, where P and Q are predicates of arity 0. Let (I_1,I_2) be the three-valued interpretation with $I_1(t) = \emptyset$ and $I_2(t) = \{Q\}$ for all $t \in \mathbb{T}$. Clearly, I_1 is a model for the empty dataset $\mathscr{D}$. Moreover, since the antecedent in both conditions of Definition 5.1 are not satisfied, (I_1,I_2) is an HT-model for $\mathscr{D}$ and Π. However, (I_1,I_2) is not a three-valued model: $[\![\mathsf{not}\ Q]\!](I_1,I_2,t) = [\![Q]\!](I_1,I_2,t)^{-1} = ([\![Q]\!](I_1,t),[\![Q]\!](I_2,t))^{-1} = \mathit{undef}^{-1} = \mathit{undef} \not\leq_\tau \mathit{false} = [\![P]\!](I_1,I_2,t)$, for all $t \in \mathbb{T}$.*

We now focus on *stable HT-models*, and show that they correspond to our two-valued stable models.

Definition 5.2. *Let $\mathscr{D}$ be a dataset, Π be a program, and $I \in \mathbb{I}$. I is a* stable HT-model of $\mathscr{D}$ and Π *if (I,I) is an HT-model of $\mathscr{D}$ and Π and*

$$I = \bigsqcap_{\subseteq} \{J \mid (J,I) \text{ is an HT-model of } \mathscr{D} \text{ and } \Pi\}. \tag{3}$$

Lemma 5.1. *Let $\mathscr{D}$ be a dataset, Π be a program, and $I,J \in \mathbb{I}$. If (I,J) is an HT-model, then I is a prefixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,J)$.*

Proof. We have to show that $\Psi^1_{\mathscr{D},\Pi}(I,J) \subseteq I$. Let $t \in \mathbb{T}$, and $P(\mathbf{s}) \in \Psi^1_{\mathscr{D},\Pi}(I,J)(t)$. Then, there are two possible cases:

1. There exists an interval δ such that $t \in \delta$ and $P(\mathbf{s})@\delta \in \mathscr{D}$. In this case, it must hold that $P(\mathbf{s}) \in I(t)$, because (I,J) is an HT-model of $\mathscr{D}$ and Π.
2. There exist $M \leftarrow B \in \mathsf{ground}(\Pi)$ and $t' \in \mathbb{T}$ such that $[\![B]\!](I,J,t') = \mathit{true}$ and $(P(\mathbf{s}),t) \in F(M,t')$. By Definition 3.1, it is easy to see that $[\![B]\!](I,J,t') = \mathit{true}$ satisfies the antecedent of Item 1 of Definition 5.1. Since (I,J) is an HT-model, we can derive that $[\![M]\!](I,t') = \mathit{true}$. By Proposition 3.1, we have that $[\![P(\mathbf{s})]\!](I,t) = \mathit{true}$, i.e. $P(\mathbf{s}) \in I(t)$.

In both cases above we can conclude that $P(\mathbf{s}) \in I(t)$. By the arbitrarity of $P(\mathbf{s})$ and t, it holds that $\Psi^1_{\mathscr{D},\Pi}(I,J) \subseteq I$, i.e. I is a prefixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,J)$, as desired. □

Theorem 5.1. *Let $\mathscr{D}$ be a dataset, Π be a program, and $I \in \mathbb{I}$. I is a stable HT-model of $\mathscr{D}$ and Π if and only if I is a two-valued stable model of $\mathscr{D}$ and Π.*

Proof. Suppose I is a stable HT-model. By Definition 5.1, I is a model of $\mathscr{D}$. We now prove that (I,I) is a stable fixpoint of $\Psi_{\mathscr{D},\Pi}$, i.e., $I = \mathrm{lfp}\,\Psi^1_{\mathscr{D},\Pi}(\cdot,I) = \mathrm{lfp}\,\Psi^2_{\mathscr{D},\Pi}(I,\cdot)$. By Theorem 4.1, $\Psi^1_{\mathscr{D},\Pi}(I,I) = I$ if and only if $\Psi^2_{\mathscr{D},\Pi}(I,I) = I$, i.e., I is a fixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$ if and only if I is a fixpoint of $\Psi^2_{\mathscr{D},\Pi}(I,\cdot)$. Since we consider only consistent pairs, i.e., pairs (J_1,J_2) with $J_1 \subseteq J_2$, it is sufficient to prove that $I = \mathrm{lfp}\,\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$. By Lemma 5.1, I is a pre-fixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$. We now show that $I \subseteq \Psi^1_{\mathscr{D},\Pi}(I,I)$.

Since I is a stable HT-model, it suffices to prove that $(\Psi^1_{\mathscr{D},\Pi}(I,I),I)$ is an HT-model of $\mathscr{D}$ and Π. Clearly, $\Psi^1_{\mathscr{D},\Pi}(I,I)$ is a model of $\mathscr{D}$ by Definition 3.4. It remains to verify the two conditions listed in Definition 5.1. Let $B = M_1 \wedge \ldots \wedge M_k \wedge \mathsf{not}\ M_{k+1} \wedge \ldots \wedge \mathsf{not}\ M_m$ such that $M \leftarrow B \in \mathsf{ground}(\Pi)$, and $t \in \mathbb{T}$:

1. Suppose $[\![M_i]\!](\Psi^1_{\mathscr{D},\Pi}(I,I),t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](I,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. By Lemma 4.1, it holds that $[\![M_i]\!](I,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$. Clearly, since all the M_i's do not contain negation, we also have that $[\![M_i]\!](I,I,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](I,I,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. Hence, we have that $[\![B]\!](I,I,t) = \mathit{true}$. By Definition 3.4, $P(\mathbf{s}) \in \Psi^1_{\mathscr{D},\Pi}(I,I)(t')$ for all $t' \in \mathbb{T}$ and $P(\mathbf{s})$ ground relational atoms such that $(P(\mathbf{s}),t') \in F(M,t)$. By Proposition 3.1, it follows that $[\![M]\!](\Psi^1_{\mathscr{D},\Pi}(I,I),t) = \mathit{true}$, as desired.

2. Suppose $[\![M_i]\!](I,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](I,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. By the same reasoning performed at the point above, it follows that $[\![M]\!](\Psi^1_{\mathscr{D},\Pi}(I,I),t) = \mathit{true}$. By Lemma 4.1, we get that $[\![M]\!](I,t) = \mathit{true}$, as desired

Thus, we can conclude that $(\Psi^1_{\mathscr{D},\Pi}(I,I),I)$ is an HT-model. Since I is a stable HT-model, it follows that $\Psi^1_{\mathscr{D},\Pi}(I,I) = I$, i.e. I is a fixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$.

We prove now that I is the *least* fixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$. Let $J \subseteq I$ be a fixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$. It suffices to show that $I \subseteq J$. If we show that (J,I) is an HT-model, we can again conclude by the minimality of I. Hence, we proceed to show that (J,I) is indeed an HT-model of $\mathscr{D}$ and Π. First, observe that since $J = \Psi^1_{\mathscr{D},\Pi}(J,I)$, J is a model of $\mathscr{D}$ by Definition 3.4. It remains to verify the two conditions listed in Definition 5.1. Let $B = M_1 \wedge \ldots \wedge M_k \wedge \mathsf{not}\ M_{k+1} \wedge \ldots \wedge \mathsf{not}\ M_m$ such that $M \leftarrow B \in \mathsf{ground}(\Pi)$, and $t \in \mathbb{T}$:

1. Suppose $[\![M_i]\!](J,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](I,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. By Lemma 4.1, it holds that $[\![M_i]\!](I,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$, and $[\![M_j]\!](J,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. Hence, we have that $[\![M_i]\!](J,I,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](J,I,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. It follows that $[\![B]\!](J,I,t) = \mathit{true}$. By Definition 3.4, $P(\mathbf{s}) \in \Psi^1_{\mathscr{D},\Pi}(J,I)(t') = J(t')$ for all $t' \in \mathbb{T}$ and $P(\mathbf{s})$ ground relational atoms such that $(P(\mathbf{s}),t') \in F(M,t)$. By Proposition 3.1, it follows that $[\![M]\!](J,t) = \mathit{true}$, as desired.

2. Suppose $[\![M_i]\!](I,t) = \mathit{true}$ for all $i \in \{1,\ldots,k\}$ and $[\![M_j]\!](I,t) = \mathit{false}$ for all $j \in \{k+1,\ldots,m\}$. By the same reasoning performed at the point above, it follows that $[\![M]\!](J,t) = \mathit{true}$. By Lemma 4.1, we get that $[\![M]\!](I,t) = \mathit{true}$, as desired.

Thus, we can conclude that (J,I) is an HT-model. Since I is a stable HT-model, it follows that $J = I$, i.e. I is the least fixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$.

Assume now that (I,I) is a three-valued stable model of $\mathscr{D}$ and Π. By Definition 4.1, it holds that I is a fixpoint of both $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$ and $\Psi^2_{\mathscr{D},\Pi}(I,\cdot)$, i.e. $\Psi_{\mathscr{D},\Pi}(I,I) = (I,I)$. By Proposition 5.1, (I,I) is an HT-model of $\mathscr{D}$ and Π. It remains to show that the equality (3) in Definition 5.2 holds. Let $J \subseteq I$ such that (J,I) is an HT-model of $\mathscr{D}$ and Π. We show that indeed J must be equal to I. It suffices to prove that $I \subseteq J$.

By Proposition 3.3 of [13], $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$ is a monotone operator on the complete lattice $\langle \{H \mid H \subseteq I\}, \subseteq \rangle$. Hence, by the Tarski–Knaster Theorem [27], the least fixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$ coincide with its least pre-fixpoint. Since $\mathrm{lfp}\,\Psi^1_{\mathscr{D},\Pi}(\cdot,I) = I$ by hypothesis, it follows that I is also the least pre-fixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$. By Lemma 5.1, J is a prefixpoint of $\Psi^1_{\mathscr{D},\Pi}(\cdot,I)$. Thus, it must be the case that $I \subseteq J$, as desired. □

6 Conclusions and Future Work

Using the solid framework of Approximation Fixpoint Theory, we have defined several semantics of interest for DatalogMTL¬: the Kripke-Kleene, well-founded, stable, and supported model semantics. This was possible by defining a *consistent approximator*, i.e. an operator on the space of partial, three-valued interpretations. As we have shown, such an operator does not require any farfetched choice; on the contrary, it is the most natural adaptation of Fitting's three-valued immediate consequence operator to the setting of DatalogMTL¬, showcasing, yet again, the simplicity, versatility, and power of the AFT formalism. In the final section, we have also proved that the stable models obtained via AFT are equivalent to the ones defined in [29] through interpretations stemming from the logic of HT.

Since pairs of interpretations are used both in AFT and for the logic of HT, although in a different way, it is often the case that when the logic of HT is employed to define some *stable models*, AFT can provide an easier route to achieve the same results, while also giving access to other types of models for free. This might be true for the *metric stable models* defined in [3], used to obtain the non-monotonic Metric Equilibrium Logic (MEL) starting from the Metric logic of HT, a temporal extension of the logic of HT. If this is indeed the case, AFT could provide a solid and reliable ground for a comparison between MEL developed in [3], and the extension of Datalog MTL with negation-as-failure of [29], discussed in this paper.

It is also worth mentioning that AFT grants access to a wide body of theoretical results that hold for its applications. A line of work focused on stratification results [28, 7]. In particular, a notion of *stratifiable operator* was defined, allowing for the computation of the Kripke-Kleene, supported, well-founded, and stable models stratum by stratum. This seems to be equivalent to what was done for stratifiable DatalogMTL¬, i.e. for DatalogMTL¬ programs with no cyclic dependencies via negation [10].

References

[1] S. Abiteboul, R. Hull & V. Vianu (1995): *Foundations of Databases*. Addison-Wesley.

[2] C. Antic (2020): *Fixed point semantics for stream reasoning*. *Artif. Intell.* 288, p. 103370, doi:10.1016/J.ARTINT.2020.103370.

[3] A. Becker, P. Cabalar, M. Diéguez, T. Schaub & A. Schuhmann (2024): *Metric Temporal Equilibrium Logic over Timed Traces*. *Theory Pract. Log. Program.* 24(3), pp. 425–452, doi:10.1017/S1471068424000139.

[4] B. Bogaerts (2019): *Weighted Abstract Dialectical Frameworks through the Lens of Approximation Fixpoint Theory*. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, AAAI Press, pp. 2686–2693, doi:10.1609/AAAI.V33I01.33012686.

[5] B. Bogaerts & L. Cruz-Filipe (2018): *Fixpoint semantics for active integrity constraints*. *Artif. Intell.* 255, pp. 43–70, doi:10.1016/J.ARTINT.2017.11.003.

[6] B. Bogaerts & M. Jakubowski (2021): *Fixpoint Semantics for Recursive SHACL*. In A. Formisano, Y. A. Liu, B. Bogaerts, A. Brik, V. Dahl, C. Dodaro, P. Fodor, G. L. Pozzato, J. Vennekens & N. Zhou, editors: *Proceedings 37th International Conference on Logic Programming (Technical Communications), ICLP Technical Communications 2021, Porto (virtual event), 20-27th September 2021*, *EPTCS* 345, pp. 41–47, doi:10.4204/EPTCS.345.14.

[7] Bart Bogaerts & Luís Cruz-Filipe (2021): *Stratification in Approximation Fixpoint Theory and Its Application to Active Integrity Constraints*. *ACM Trans. Comput. Log.* 22(1), pp. 6:1–6:19, doi:10.1145/3430750.

[8] S. Brandt, E. G. Kalayci, V. Ryzhikov, G. Xiao & M. Zakharyaschev (2018): *Querying Log Data with Metric Temporal Logic*. *J. Artif. Intell. Res.* 62, pp. 829–877, doi:10.1613/JAIR.1.11229.

[9] A. Colombo, L. Bellomarini, S. Ceri & E. Laurenza (2023): *Smart Derivative Contracts in DatalogMTL*. In J. Stoyanovich, J. Teubner, N. Mamoulis, E. Pitoura, J. Mühlig, K. Hose, S. S. Bhowmick & M. Lissandrini, editors: *Proceedings 26th International Conference on Extending Database Technology, EDBT 2023, Ioannina, Greece, March 28-31, 2023*, OpenProceedings.org, pp. 773–781, doi:10.48786/EDBT.2023.65.

[10] D. J. Tena Cucala, P. A. Walega, B. Cuenca Grau & E.V. Kostylev (2021): *Stratified Negation in Datalog with Metric Temporal Operators*. In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, AAAI Press, pp. 6488–6495, doi:10.1609/AAAI.V35I7.16804.

[11] M. Denecker, M. Bruynooghe & J. Vennekens (2012): *Approximation Fixpoint Theory and the Semantics of Logic and Answers Set Programs*. In E. Erdem, J. Lee, Y. Lierler & D. Pearce, editors: *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, *Lecture Notes in Computer Science* 7265, Springer, pp. 178–194, doi:10.1007/978-3-642-30743-0_13.

[12] M. Denecker, V. Marek & M. Truszczyński (2000): *Approximations, Stable Operators, Well-Founded Fixpoints and Applications in Nonmonotonic Reasoning*. In: *Logic-Based Artificial Intelligence*, 597, Springer US, pp. 127–144, doi:10.1007/978-1-4615-1567-8_6.

[13] M. Denecker, V. W. Marek & M. Truszczyński (2004): *Ultimate approximation and its application in nonmonotonic knowledge representation systems*. *Inf. Comput.* 192(1), pp. 84–121, doi:10.1016/J.IC.2004.02.004.

[14] M. Fitting (2002): *Fixpoint semantics for logic programming a survey*. *Theor. Comput. Sci.* 278(1-2), pp. 25–51, doi:10.1016/S0304-3975(00)00330-3.

[15] A. Heyting (1930): *Die formalen Regeln der intuitionistischen Logik*. In: *Sitzungsberichte der Preussischen Akademie der Wissenschaften, physikalisch-mathematische klass.*

[16] S. S. Huang, T. J. Green & B. T. Loo (2011): *Datalog and emerging applications: an interactive tutorial*. In T. K. Sellis, R. J. Miller, A. Kementsietsidis & Y. Velegrakis, editors: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, ACM, pp. 1213–1216, doi:10.1145/1989323.1989456.

[17] E. G. Kalayci, G. Xiao, V. Ryzhikov, T. E. Kalayci & D. Calvanese (2018): *Ontop-temporal: A Tool for Ontology-based Query Answering over Temporal Data*. In A. Cuzzocrea, J. Allan, N. W. Paton, D. Srivastava, R. Agrawal, A. Z. Broder, M. J. Zaki, K. S. Candan, A. Labrinidis, A. Schuster & H. Wang, editors: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, ACM, pp. 1927–1930, doi:10.1145/3269206.3269230.

[18] B. Ketsman & P. Koutris (2022): *Modern Datalog Engines*. *Found. Trends Databases* 12(1), pp. 1–68, doi:10.1561/1900000073.

[19] S. Kikot, V. Ryzhikov, P. A. Walega & M. Zakharyaschev (2018): *On the Data Complexity of Ontology-Mediated Queries with MTL Operators over Timed Words*. In M. Ortiz & T. Schneider, editors: *Proceedings of the 31st International Workshop on Description Logics co-located with 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018), Tempe, Arizona, US, October 27th - to - 29th, 2018*, *CEUR Workshop Proceedings* 2211, CEUR-WS.org.

[20] P. Koopmann (2019): *Ontology-Based Query Answering for Probabilistic Temporal Data*. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, AAAI Press, pp. 2903–2910, doi:10.1609/AAAI.V33I01.33012903.

[21] R. Koymans (1990): *Specifying Real-Time Properties with Metric Temporal Logic*. *Real Time Syst.* 2(4), pp. 255–299, doi:10.1007/BF01995674.

[22] M. Mori, P. Papotti, L. Bellomarini & O. Giudice (2022): *Neural Machine Translation for Fact-checking Temporal Claims*. In R. Aly, C. Christodoulopoulos, O. Cocarascu, Z. Guo, A. Mittal, M. Schlichtkrull, J. Thorne & A. Vlachos, editors: *Proceedings of the Fifth Fact Extraction and VERification Workshop (FEVER)*, Association for Computational Linguistics, Dublin, Ireland, pp. 78–82, doi:10.18653/v1/2022.fever-1.8.

[23] M. Nissl & E. Sallinger (2022): *Modelling Smart Contracts with DatalogMTL*. In M. Ramanath & T. Palpanas, editors: *Proceedings of the Workshops of the EDBT/ICDT 2022 Joint Conference, Edinburgh, UK, March 29, 2022*, *CEUR Workshop Proceedings* 3135, CEUR-WS.org.

[24] D. Pearce (1996): *A New Logical Characterisation of Stable Models and Answer Sets*. In J. Dix, L. Moniz Pereira & T. C. Przymusinski, editors: *Non-Monotonic Extensions of Logic Programming, NMELP '96, Bad Honnef, Germany, September 5-6, 1996, Selected Papers*, *Lecture Notes in Computer Science* 1216, Springer, pp. 57–70, doi:10.1007/BFB0023801.

[25] S. Pollaci, B. Kostopoulos, M. Denecker & B. Bogaerts (2025): *A Category-Theoretic Perspective on Higher-Order Approximation Fixpoint Theory*. In C. Dodaro, G. Gupta & M. V. Martinez, editors: *Logic Programming and Nonmonotonic Reasoning*, *Lecture Notes in Computer Science* 15245, Springer Nature Switzerland, Cham, pp. 317–330, doi:10.1007/978-3-031-74209-5_24.

[26] H. Strass (2013): *Approximating operators and semantics for abstract dialectical frameworks*. *Artif. Intell.* 205, pp. 39–70, doi:10.1016/J.ARTINT.2013.09.004.

[27] A. Tarski (1955): *A lattice-theoretical fixpoint theorem and its applications*. *Pacific journal of mathematics* 5, pp. 285–309, doi:10.2140/pjm.1955.5.285.

[28] J. Vennekens, D. Gilis & M. Denecker (2006): *Splitting an operator: Algebraic modularity results for logics with fixpoint semantics*. *ACM Trans. Comput. Log.* 7(4), pp. 765–797, doi:10.1145/1183278.1183284.

[29] P. A. Walega, D. J. Tena Cucala, B. Cuenca Grau & E. V. Kostylev (2024): *The Stable Model Semantics of Datalog with Metric Temporal Operators*. *Theory Pract. Log. Program.* 24(1), pp. 22–56, doi:10.1017/S1471068423000315.

[30] P. A. Walega, M. Kaminski & B. C. Grau (2019): *Reasoning over Streaming Data in Metric Temporal Datalog*. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, AAAI Press, pp. 3092–3099, doi:10.1609/AAAI.V33I01.33013092.

[31] P. A. Walega, M. Kaminski, D. Wang & B. C. Grau (2023): *Stream reasoning with DatalogMTL*. *J. Web Semant.* 76, doi:10.1016/J.WEBSEM.2023.100776.

A Refined Operational Semantics for FreeCHR

Sascha Rechenberger and Thom Frühwirth
Institute for Software Engineering and Programming Languages
Ulm University
Albert-Einstein-Allee 11, 89069 Ulm, Germany
{sascha.rechenberger,thom.fruehwirth}@uni-ulm.de

Constraint Handling Rules (CHR) is a rule-based programming language which is typically embedded into a general-purpose language. There exists a plethora of implementations for numerous host languages. However, the existing implementations often re-invent the way to embed CHR, which impedes maintenance and weakens assertions of correctness. To formalize and thereby standardize the embedding into arbitrary host languages, we introduced the framework *FreeCHR* and proved it to be a valid representation of classical ground CHR. Until now, this framework only includes a translation of the *very abstract* operational semantics which, due to its abstract nature, is not a sufficient base for practical implementations. In this paper we present a translation of the *refined* operational semantics for FreeCHR and prove it to be both a valid concretization of the *very abstract* semantics of FreeCHR, and an equivalent representation of the *refined* semantics of CHR. This will establish implementations of FreeCHR as equivalent in behavior and expressiveness to existing implementations of CHR.

1 Introduction

Constraint Handling Rules (CHR) is a rule-based programming language which is typically embedded into a general-purpose language. Having a CHR implementation available enables software developers to solve problems in a declarative and elegant manner. Aside from the obvious task of implementing constraint solvers [10, 7], it has been used, e.g. to solve scheduling problems [3] and implement concurrent and multi-agent systems [27, 28, 18, 19]. In general, CHR is ideally suited for any problem that involves the transformation of collections of data. Programs consist of rewriting rules which hide the process of finding matching values to which rules can be applied. As a result, developers are able to implement algorithms and applications in a purely declarative way, without the otherwise necessary boilerplate code.

The literature on CHR as a formalism consists of a rich body of theoretical work, including a rigorous formalization of its declarative and operational semantics [11, 26, 12], relations to other rule-based formalisms [13] and results on properties like confluence [1, 6, 14]. Implementations of CHR exist for a number of languages, such as Prolog [25], C [32], C++ [4], Haskell [5, 19], JavaScript [20] and Java [2, 30, 16, 31].

We intend to formalize and thereby standardize the embedding of CHR into arbitrary host languages as an internal DSL. In our view, this will have three major advantages. First, by formalizing CHR via *initial algebra semantics*, we provide a tighter connection between the formal definition and the implementations. Initial algebra semantics are a common concept in functional programming and are used to inductively define and implement languages and their semantics [15, 17]. This provides a common formal foundation for considerations concerning theory and implementations.

Secondly, by embedding CHR as an internal language, distribution and use are greatly increased. Unlike embeddings which rely on an external source-to-source compiler, internal embeddings can be

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 278–293, doi:10.4204/EPTCS.439.20

© S. Rechenberger, T. Frühwirth
This work is licensed under the
Creative Commons Attribution License.

easily distributed and imported via the host languages package and module system. They also do not add another link to the build chain which is, even with modern build tools, an additional nuisance.

Finally, by providing formal, host language agnostic and verifiable definitions of a CHR implementation, we also provide a generalized documentation for implementations. We hope that this will ease contribution to (in future) existing implementations and help to keep them maintained.

We hence introduced the framework *FreeCHR* [21]. It provides both, a guideline and high-level architecture to implement and maintain CHR implementations across host languages, and a strong connection between the practical and formal aspects of CHR. By using initial algebra semantics, we also get a formalization for type-safe programs as well as composition of programs for free.

However, the framework currently only formalizes the *very abstract* operational semantics of CHR which are too abstract to serve as basis for standardized efficient implementations. The *refined* operational semantics ω_r, introduced by Duck et al. [9], formalizes the behavior of efficient existing CHR systems. It fixes the order in which rules are tried to be applied as well as the order in which the head of a rule is traversed in order to find suitable values for rule application. Hereby, the programmer can use the control flow of the CHR system to write more efficient programs. The refined semantics also prevents reapplication of rules with the same applicable values. This fixes the issue of trivial non-termination of certain types of programs.

In this paper we hence will introduce the *refined* semantics to FreeCHR. We will provide a modified definition $\omega_r^\star$ and prove their soundness and completeness w.r.t. ω_r. We will also prove soundness w.r.t. the existing *very abstract* operational semantics to guarantee that all definitions within our framework are consistent. The operational semantics presented herein are already implemented in our ongoing work [22, 23].

The rest of the paper is structured as follows: Section 2 introduces necessary preliminary notations and definitions, Section 3 the definition of ground *Constraint Handling Rules* over non-Herbrand domains and the *refined* semantics for CHR, and Section 4 the definition of *FreeCHR* and its *very abstract* semantics. The remaining sections contain our new contributions. Section 5 introduces the *refined* semantics of FreeCHR, Section 6 proves their soundness w.r.t. the *very abstract* semantics of FreeCHR and Section 7 proves their soundness and completeness w.r.t. the *refined* semantics of CHR. Finally Section 8 concludes the paper. An extended preprint of this paper, including full proofs, is available at `arXiv.com` [24].

2 Preliminaries

We first want to define the basic concepts and notations we use throughout the paper.

2.1 Basic notations

The *disjoint union* $A \sqcup B = \{l_A(a) \mid a \in A\} \cup \{l_B(b) \mid b \in B\}$ of two sets A and B is the union of both sets, with additional labels l_A and l_B added to the elements, to keep track of the origin set of each element. We will also use the labels l_A and l_B as *injection* functions $l_A : A \to A \sqcup B$ and $l_B : B \to A \sqcup B$ which construct elements of $A \sqcup B$ from elements of A or B, respectively. For two functions $f : A \to C$ and $g : B \to C$, the function $[f,g] : A \sqcup B \to C$ is defined as

$$[f,g](l(x)) = \begin{cases} f(x), & \text{if } l = l_A \\ g(x), & \text{if } l = l_B \end{cases}$$

It can be understood as a formal analogue to a `case ... of` expression. Furthermore, we define two functions $f \sqcup g : A \sqcup B \to A' \sqcup B'$ and $f \times g : A \times B \to A' \times B'$ as

$$(f \sqcup g)(l(x)) = \begin{cases} l_{A'}(f(x)), & \text{if } l = l_A \\ l_{B'}(g(x)), & \text{if } l = l_B \end{cases} \qquad (f \times g)(x,y) = (f(x), g(y))$$

which lift two functions $f : A \to A'$ and $g : B \to B'$ to the disjoint union or the Cartesian product, respectively.

2.2 Functors and F-algebras

A functor F maps all sets A to sets FA and all functions $f : A \to B$ to functions $Ff : FA \to FB$, such that $F\,\mathbf{id}_A = \mathbf{id}_{FA}$ and $F(g \circ f) = Fg \circ Ff$. $\mathbf{id}_X(x) = x$ is the identity function on a set X. A signature $\Sigma = \{\sigma_1/a_1, ..., \sigma_n/a_n\}$, where σ_i are operators and a_i their arity, generates a functor F_Σ with

$$F_\Sigma X = \bigsqcup_{\sigma/a \in \Sigma} X^a \qquad F_\Sigma f = \bigsqcup_{\sigma/a \in \Sigma} f^a$$

X^n and f^n are defined as

$$X^n = \underbrace{X \times ... \times X}_{n \text{ times}} \qquad f^n = \underbrace{f \times ... \times f}_{n \text{ times}}$$

and $X^0 = \mathbb{1}$ and $f^0 = \mathbf{id}_\mathbb{1}$, where $\mathbb{1}$ is a singleton set. Such a functor F_Σ models *flat* (i.e., not nested) terms over the signature Σ.

Since an endofunctor F defines the syntax of terms, an evaluation function $\alpha : FA \to A$ defines the *semantics* of terms. We call such a function α, together with its *carrier* A, an F-algebra (A, α). If there are two F-algebras (A, α) and (B, β) and a function $h : A \to B$, we call h an *F-algebra homomorphism*, iff. $h \circ \alpha = \beta \circ Fh$, i.e., h preserves the structure of (A, α) in (B, β) when mapping A to B. In this case, we also write $h : (A, \alpha) \to (B, \beta)$.

A special F-algebra is the *free F-algebra* $F^\star = (\mu F, \mathbf{in}_F)$, for which there is a homomorphism $(\!|\alpha|\!) : F^\star \to (A, \alpha)$ for any other algebra (A, α). We call those homomorphisms $(\!|\alpha|\!)$ *F-algebra catamorphisms*. The functions $(\!|\alpha|\!)$ encapsulate structured recursion on values in μF with the semantics defined by the function α which is itself only defined on flat terms. The carrier of $F^\star$, with $\mu F = F\mu F$, is the set of inductively defined values in the shape defined by F. The function $\mathbf{in}_F : F\mu F \to \mu F$ inductively constructs the values in μF.

2.3 Labelled transition systems

A *labelled transition system* (LTS) $\omega = \langle \Sigma, L, (\mapsto) \rangle$ consists of a set Σ called the *domain*, a set L called *actions* and a ternary *transition relation* $(\mapsto) \subseteq \Sigma \times L \times \Sigma$. The idea is that if $s \overset{l}{\longmapsto} s' \in (\mapsto)$, we transition from state s to s' by the action l.

For two LTS $\omega_1 = \langle \Sigma_1, L_1, (\mapsto) \rangle$ and $\omega_2 = \langle \Sigma_2, L_2, (\hookrightarrow) \rangle$ and two functions $f : \Sigma_1 \longrightarrow \Sigma_2$ and $g : L_1 \longrightarrow L_2$ we say that ω_1 is (f,g)-sound w.r.t. ω_2, iff.

$$s \overset{l}{\longmapsto} s' \in (\mapsto) \Longrightarrow f(s) \overset{g(l)}{\hookrightarrow} f(s') \in (\hookrightarrow) \qquad ((f,g)\text{-soundness})$$

and (f,g)-complete w.r.t. ω_2, iff.

$$s \overset{l}{\longmapsto} s' \in (\mapsto) \Longleftarrow f(s) \overset{g(l)}{\longrightarrow} f(s') \in (\hookrightarrow) \qquad ((f,g)\text{-completeness})$$

By $(\mapsto)^+$ we denote the *transitive* and by $(\mapsto)^*$ the *reflexive-transitive* closure of $(\mapsto)$. Recall that $(\mapsto)^+ \subset (\mapsto)^*$, for every $R \subseteq (\mapsto)^+$, $R^+ \subseteq (\mapsto)^+$ and for every $Q \subseteq (\mapsto)^*$, $Q^* \subseteq (\mapsto)^*$.

3 Ground CHR over non-Herbrand domains

In this section, we will reiterate definitions concerning the generalization of CHR to non-Herbrand domains and introduce its *refined* operational semantics.

A *data type* determines the syntax and semantics of terms via a functor Λ_T and a Λ_T-algebra τ_T. The fixed point $\mu\Lambda_T$ contains terms which are inductively defined via Λ_T and the Λ_T-catamorphism $(\!|\tau_T|\!)$ evaluates those terms to values of T.

Definition 1 (Data types)**.** A *data type* is a triple $\langle T, \Lambda_T, \tau_T \rangle$, where T is a set, Λ_T a functor and (T, τ_T) a Λ_T-algebra.

We write $t \equiv_T t'$ for $t \in \mu\Lambda_T$ and $t' \in T$, iff. $(\!|\tau_T|\!)(t) = t'$.

Example 1 (Boolean data type). The signature

$$\Sigma_{\mathbb{2}} = \{(n \leq m)/0 \mid n,m \in \mathbb{N}_0\} \cup \{(n < m)/0 \mid n,m \in \mathbb{N}_0\} \cup \{\wedge/2, \mathit{true}/0, \mathit{false}/0\}$$

defines Boolean terms[1]. $\Sigma_{\mathbb{2}}$ generates the functor

$$\Lambda_{\mathbb{2}} X = \mathbb{N}_0 \times \mathbb{N}_0 \sqcup\ \mathbb{N}_0 \times \mathbb{N}_0 \sqcup\ X \times X \sqcup \mathbb{1} \sqcup \mathbb{1}$$

the fixed point, $\mu\Lambda_{\mathbb{2}}$, of which is the set of valid nested Boolean terms like $(0 < 4 \wedge 4 \leq 6)$. Let $\langle \mathbb{2}, \Lambda_{\mathbb{2}}, \tau_{\mathbb{2}} \rangle$, with $\mathbb{2} = \{\mathit{true}, \mathit{false}\}$, be a data type. If we assume $\tau_{\mathbb{2}}$ to implement the usual semantics for Boolean terms and comparisons, $(0 < 4 \wedge 4 \leq 6)$ will evaluate as

$$(\!|\tau_{\mathbb{2}}|\!)(0 < 4 \wedge 4 \leq 6) = (\!|\tau_{\mathbb{2}}|\!)(0 < 4) \wedge (\!|\tau_{\mathbb{2}}|\!)(4 \leq 6) = \mathit{true} \wedge \mathit{true} = \mathit{true}$$

We can therefor write $0 < 4 \wedge 4 \leq 6 \equiv_{\mathbb{2}} \mathit{true}$.

For a set T, both Λ_T and τ_T are determined by the host language which is captured by the next definition.

Definition 2 (Host environment)**.** A mapping $\mathscr{L}T = \langle T, \Lambda_T, \tau_T \rangle$, where $\langle T, \Lambda_T, \tau_T \rangle$ is a data type, is called a *host environment*.

We assume that a host environment is provided by the host language (and the program, the CHR program is part of) and assigns a data type to a set T, effectively determining syntax and semantics of terms that evaluate to values of T.

Definition 3 (Non-Herbrand ground CHR programs)**.** CHR programs are sequences of multiset-rewriting rules of the generalized form

$$N \mathbin{@} K \setminus R \iff [G \mid] B$$

[1] We will generally overload symbols like *false*, *true*, $\wedge$, $\neg$, ..., if their meaning is clear from the context.

N is the identifier of the rule. It has to be unique in the executed program. For a set C, called the *domain* of the Program, for which there is a data type $\mathscr{L}C = \langle C, \Lambda_C, \tau_C \rangle$, $K, R \in \texttt{list}\, C$ are called the *kept* and *removed head*, respectively. Either can be omitted, but not both at the same time. If K is empty, we call the rule a *simplification* rule, and if R is empty, a *propagation* rule. If both K and R are non-empty, the rule is called a *simpagation* rule. The functor $\texttt{list}\, X = \bigcup_{i \in \mathbb{N}} X^i$ maps a set X to the set of finite sequences (or lists) over X, with $X^0 = []$ being the empty sequence[2]. The optional $G \in \mu\Lambda_{\mathbb{2}}$ is called the *guard*. If G is omitted, we assume $G \equiv_{\mathbb{2}} \mathit{true}$. $B \in \texttt{list}\, \mu\Lambda_C$ is called the *body*.

The members of the kept and removed head are matched against values of the domain C. The guard G is a term that can be evaluated to a Boolean value in $\mathbb{2}$. The body B is a list of terms which can be evaluated to values of C. This includes any call of (pure) functions or operators which evaluates to Booleans, or values of C, respectively.

Definition 3 corresponds to the *positive range-restricted ground* segment of CHR which is commonly used as the target for embeddings of other (rule-based) formalisms [13, Chapter 7]. $\mathbf{PRG}_C$ denotes the set of all such programs over a domain C.

Example 2 (Euclidean algorithm). The program GCD $= [\mathit{zero}@..., \mathit{subtract}@...]$

$$\begin{aligned} \mathit{zero} \ @ \ 0 &\iff \emptyset \\ \mathit{subtract} \ @ \ N \setminus M &\iff 0 < N \wedge 0 < M \wedge N \leq M \mid M - N \end{aligned}$$

computes the greatest common divisor of a collection of natural numbers. The *zero* rule removes any occurrences of 0 from the collection. The *subtract* rule replaces for any pair of numbers N and M greater 0 and $N \leq M$, M by $M - N$.

Definition 4 (C-groundings & instances of rules)**.** For a *positive range-restricted ground* rule

$$r = R \ @ \ k_1, ..., k_n \setminus r_1, ..., r_m \Leftrightarrow G \mid B$$

with universally quantified variables $v_1, ..., v_l$, and a data type $\mathscr{L}C = \langle C, \Lambda_C, \tau_C \rangle$, we call the set

$$\begin{aligned} \Gamma_C(r) = \{ \ & (R \ @ \ k_1\sigma, ..., k_n\sigma \setminus r_1\sigma, ..., r_m\sigma \Leftrightarrow G\sigma \mid \texttt{list}(\!|\tau_C|\!)(B\sigma)) \\ & \mid \text{the substitution } \sigma \text{ instantiates all variables } v_1, ..., v_l, \\ & k_1\sigma, ..., k_n\sigma, r_1\sigma, ..., r_m\sigma \in C, \\ & G\sigma \in \mu\Lambda_{\mathbb{2}}, \\ & B\sigma \in \texttt{list}\, \mu\Lambda_C \ \} \end{aligned}$$

the *C-grounding* of r. Analogously, for a program $\mathscr{R}$, $\Gamma_C(\mathscr{R}) = \bigcup_{r \in \mathscr{R}} \Gamma_C(r)$ is the C-grounding of $\mathscr{R}$. An element $r' \in \Gamma_C(r)$ (or $\Gamma_C(\mathscr{R})$ respectively) is called a *C-instance* of a rule $r \in \mathscr{R}$.

A C-instance (or grounding) is obtained, by instantiating any variables and evaluating the then ground terms in the body of the rule, using the Λ_C-catamorphism $(\!|\tau_C|\!)$.

Example 3. Given a body $[M - N]$ and a substitution $\sigma = \{N \mapsto 4, M \mapsto 6\}$, the body is instantiated like

$$\texttt{list}(\!|\tau_C|\!)([M - N]\,\sigma) = \texttt{list}(\!|\tau_C|\!)([6 - 4]) = [(\!|\tau_C|\!)(6 - 4)] = \{2\}$$

[2]We use Haskell-like syntax to denote lists or sequences. [] is the empty list and $x : xs$ constructs a list with head element x and tail xs. We will also use the notations $(a : b : c : [])$ and $[a, b, c]$ interchangeably as we consider it useful. The operator $(\diamond) : \texttt{list}\, C \times \texttt{list}\, C \to \texttt{list}\, C$ denotes list concatenation.

With Example 3, we can also easily see that if we use a data type $\mathscr{L}\mu\Lambda_C = \langle \mu\Lambda_C, \Lambda_C, \mathbf{in}_{\Lambda_C} \rangle$ we get the Herbrand interpretation of terms over C. Hence, f.i. an expression $(3+4) \in \mu\Lambda_{\mathbb{N}_0}$ is evaluated to itself, as it is the case in Prolog.

Example 4 (C-instances). If we instantiate the rule

$$subtract \ @ \ N \setminus M \ \Leftrightarrow \ 0 < N \wedge 0 < M \wedge N \leq M \mid M - N$$

with $\sigma_1 = \{N \mapsto 4, M \mapsto 6\}$ and $\sigma_2 = \{N \mapsto 0, M \mapsto 6\}$, respectively, we get the $\mathbb{N}_0$-instances

$$(subtract)\sigma_1 = subtract \ @ \ 4 \setminus 6 \ \Leftrightarrow \ 0 < 4 \wedge 0 < 6 \wedge 4 \leq 6 \mid 2$$
$$(subtract)\sigma_2 = subtract \ @ \ 0 \setminus 6 \ \Leftrightarrow \ 0 < 0 \wedge 0 < 6 \wedge 0 \leq 6 \mid 6$$

Both instances are elements of the $\mathbb{N}_0$-grounding $\Gamma_{\mathbb{N}_0}(\text{GCD})$ of the program in Example 2.

Classically, the guard G contains constraints which are defined w.r.t. a constraint theory $\mathscr{C}\mathscr{T}$. We typically write $\mathscr{C}\mathscr{T} \models G$ to denote that the guard is satisfiable w.r.t. $\mathscr{C}\mathscr{T}$. In our case, $\mathscr{C}\mathscr{T}$ is essentially defined by τ_2, as it determines the semantics of Boolean terms. We thus write $\tau_2 \models G$ iff. $G \equiv_2 true$ and $\tau_2 \models \neg G$ iff. $G \equiv_2 false$. Note that we always need a data type $\mathscr{L}2$.

Example 5 (ω_a-transitions). Given the instances from Example 4, we write

$$\tau_2 \models 0 < 4 \wedge 0 < 6 \wedge 4 \leq 6 \qquad \text{and} \qquad \tau_2 \not\models 0 < 0 \wedge 0 < 6 \wedge 0 \leq 6$$

The very abstract operational semantics ω_a of CHR operates on plain multisets of values. It describes that a rule r can be applied to a state $\{k_1, ..., k_n, r_1, ..., r_m\} \uplus \Delta s$ if there is a C-instance

$$(k_1, ..., k_n \setminus r_1, ..., r_m \iff G \mid b_1, ..., b_p) \in \Gamma_C(r)$$

with $G \equiv_2 true$. If the rule is applied, the elements $\{r_1, ..., r_m\}$ are replaced by $\{b_1, ..., b_p\}$. How the *matching* $\{k_1, ..., k_n, r_1, ..., r_m\}$ is found and which one of the applicable rules of the executed program is applied is non-deterministic [11]. The following example shows the execution of the Euclidean algorithm as a final example of the operational semantics of CHR.

Example 6 (Euclidean algorithm (cont.)). The rules of GCD are applied until exhaustion, leaving only the greatest common divisor of all numbers of the input. For an input $\{4, 6\}$, the program will perform a sequence

$$\{4,6\} \overset{\{subtract\}}{\underset{a}{\longmapsto}} \{4,2\} \overset{\{subtract\}}{\underset{a}{\longmapsto}} \{2,2\} \overset{\{subtract\}}{\underset{a}{\longmapsto}} \{2,0\} \overset{\{zero\}}{\underset{a}{\longmapsto}} \{2\}$$

of transformations.

The very abstract semantics also does not account for propagation rules which remain applicable to a state, since none of the applicable values are removed. This possibly leads to non-termination if a program contains propagation rules.

Example 7 (Transitive hull). Given the program

$$trans \ @ \ (X,Y)^{\#2}, \ (Y,Z)^{\#1} \implies X \neq Z \mid (X,Z)$$

which adds the transitive edge (X,Z) of two edges (X,Y) and (Y,Z), with $X \neq Z$.

Since reapplication of a rule to the same values is allowed under the very abstract semantics the infinite transition sequence

$$\begin{aligned}
&\{(a,b),(b,c)\}\\
&\xmapsto[a]{\{trans\}} \{(a,b),(b,c),(a,c)\}\\
&\xmapsto[a]{\{trans\}} \{(a,b),(b,c),(a,c),(a,c)\}\\
&\xmapsto[a]{\{trans\}} \{(a,b),(b,c),(a,c),(a,c),(a,c)\}\\
&\xmapsto[a]{\{trans\}} \dots
\end{aligned}$$

is perfectly valid but would practically lead to non-termination.

The refined semantics solves these issues by resolving some sources of non-determinism and guaranteeing that rules are not applied more than once with the same matching. However, it requires additional information in the states to do so.

Definition 5 (States)**.** The functor

$$\Omega_r C = \texttt{list}(C \sqcup (\mathbb{N} \times C)) \times \mathscr{P}(\mathbb{N} \times C) \times \mathscr{P}(\texttt{str} \times \texttt{list}\,\mathbb{N}) \times \mathbb{N}$$

models the set of states over values in C. For an element $\langle Q,S,H,I\rangle \in \Omega_r C$ we call Q the *query*, S the *store*, H the *propagation history* and I the *index*. $\mathscr{P}(X)$ is the powerset of a set X, $\mathbb{N}$ are the natural numbers and `str` are strings.

The *query* can be understood as an execution stack. If a value in the body of a rule is added, it is handled as if it was a procedure call, where the body of the procedure is defined by the rules of the program. The *store* is a set of identifier-value pairs. It is classically viewed as a conjunction of currently known facts. The unique identifier also serves the purpose of simulating the multiset semantics required by CHR and FreeCHR. The *propagation history* is used to prevent trivial non-termination. Finally, the *index* is a natural number that is used to generate unique identifiers.

In programs for the refined semantics, the head patterns are viewed as decorated with indices incrementing from *right to left* and *top to bottom* (in textual order) throughout the program. We call them *pattern indices*. $\mathbf{PRG}_C^{\#}$ denotes the set of programs in $\mathbf{PRG}_C$ with enumerated patterns. The function *indices* maps a program in $p \in \mathbf{PRG}_C^{\#}$ to the set of pattern indices occurring in p.

Example 8 (Greatest common divisor, enumerated). The program GCD$^{\#}$ below computes the greatest common divisor. It is the enumerated variant of the program GCD shown in Example 2. The patterns of the rule are decorated with pattern indices. *indices*(GCD$^{\#}$) returns the set $\{1,2,3\}$.

$$\begin{aligned}
&\mathit{zero}\ @\ 0^{\#1} \iff \emptyset\\
&\mathit{subtract}\ @\ N^{\#3} \setminus M^{\#2} \iff 0<N \wedge 0<M \wedge N \leq M \mid M-N
\end{aligned}$$

We now define the *refined* semantics of CHR. The original definition describes six kinds of state transitions [9, 11]. Since we want to operate on ground values only, we can ignore two of them which are only concerned with non-ground values.

Definition 6 (*Refined* operational semantics for non-Herbrand ground CHR)**.** The *refined* operational semantics for non-Herbrand ground CHR are defined as an LTS

$$\omega_r = \langle\ \Omega_r C\ ,\ \mathbf{PRG}_C^{\#}\ ,\ (\underset{r}{\mapsto})^+\ \rangle$$

where the transition relation $(\underset{r}{\mapsto}) \subseteq \Omega_r C \times \mathbf{PRG}_C^{\#} \times \Omega_r C$ is defined by the rules below.

Activate The transition

$$\langle c : Q, S, H, I \rangle \xmapsto[r]{p} \langle (I,c)^{\#1} : Q, \{(I,c)\} \uplus S, H, I+1 \rangle \tag{ACTIVATE}$$

activates a value c by introducing it to the store with a unique identifier I. On the query, the value is also decorated with the pattern index #1. This indicates that it will be tried to match it to the rightmost pattern of the first rule of the program p. The activation of a value can be understood as a procedure call where the procedure is defined by the applicable rules. We use the operator $(\uplus)$ to emphasize that the operands are disjoints sets, i.e. if $A \uplus B = C$, then $A \cup B = C$ and $A \cap B = \emptyset$.

Apply Given a C-instance

$$(N \ @ \ c_1^{\#l_1}, ..., c_n^{\#l_n} \setminus c_{n+1}^{\#l_{n+1}}, ..., c_{n+m}^{l_{n+m}} \iff G \mid B) \in \Gamma_C(p)$$

such that for $j \in \{1, ..., n+m\}$, $(i_j, c_j) \in K \uplus R$, $G \equiv_2 \mathit{true}$ and $\{(N, i_1, ..., i_{n+m})\} \notin H$, we can perform the transition

$$\langle (i_j, c_j)^{\#l_j} : Q, K \uplus R \uplus S, H, I \rangle \xmapsto[r]{p} \langle B \diamond ((i_j, c_j)^{\#l_j} : Q), K \uplus S, \{(N, i_1, ..., i_{n+m})\} \cup H, I \rangle \tag{APPLY}$$

with $K = \{(i_1, c_1), ..., (i_n, c_n)\}$ and $R = \{(i_{n+1}, c_{n+1}), ..., (i_{n+m}, c_{n+m})\}$.

We need to check if the configuration $(N, i_1, ..., i_{n+m})$ already fired to prevent reapplication. If not, we record the configuration, remove R from the store and query the values of the body, by concatenating the sequence B before the query.

Drop The transition

$$\langle (i,c)^{\#j} : Q, S, H, I \rangle \xmapsto[r]{p} \langle Q, S, H, I \rangle \tag{DROP}$$

is used if j exeeds the pattern indices of the program p, i.e. if $j \notin \mathit{indices}(p)$. This indicates that there are no more applicable rules for the currently active value. This also happens, if (i,c) was removed by the APPLY transition at some point.

Default If there is a rule $r \in p$, such that $j \in \mathit{indices}(r)$, but no C-instance of r, such that the *Apply* transition can be used, we use the transition

$$\langle (i,c)^{\#j} : Q, S, H, I \rangle \xmapsto[r]{p} \langle (i,c)^{\#j+1} : Q, S, H, I \rangle \tag{DEFAULT}$$

This transition continues the traversal with the currently active value through the program.

Example 9 (Transitive hull). Given the program from Example 7. Figure 1 shows the execution of the program with an initial query $[(a,b),(b,c)]$.

First, (a,b) gets activated and, after a few transitions, dropped, as the rule requires two values to fire. Then, (b,c) gets activated and the rule *trans* fires immediately. This queries (a,c) and adds the record $(\mathit{trans}, 1, 2)$ to the propagation history. Since there is no matching partner for (a,c) in the store, the value gets dropped after activation and two DEFAULT transitions.

Now, (b,c) is active again. Without the propagation history, the rule from above could be applied again, as both $(1,(a,b))$ and $(2,(b,c))$ are still in the store. However, since the $(\mathit{trans}, 1, 2)$ is already recorded in the propagation history, the APPLY transition can not be applied. Hence, the DEFAULT transition needs to be applied and the value is dropped ultimately.

$$
\begin{aligned}
&\langle [(a,b),(b,c)],\emptyset,\emptyset,1\rangle \\
&\xmapsto[r]{[trans]} \langle [(1,(a,b))^{\#1},(b,c)],\{(1,(a,b))\},\emptyset,2\rangle && \text{(ACTIVATE)}\\
&\xmapsto[r]{[trans]}{}^{*} \langle [(1,(a,b))^{\#3},(b,c)],\{(1,(a,b))\},\emptyset,2\rangle && (2\times\text{DEFAULT})\\
&\xmapsto[r]{[trans]} \langle [(b,c)],\{(1,(a,b))\},\emptyset,2\rangle && \text{(DROP)}\\
&\xmapsto[r]{[trans]} \langle [(2,(b,c))^{\#1}],\{(1,(a,b)),(2,(b,c))\},\emptyset,3\rangle && \text{(ACTIVATE)}\\
&\xmapsto[r]{[trans]} \langle [(a,c),(2,(b,c))^{\#1}],\{(1,(a,b)),(2,(b,c))\},\{(trans,1,2)\},3\rangle && \text{(APPLY)}\\
&\xmapsto[r]{[trans]} \langle [(3,(a,c))^{\#1},(2,(b,c))^{\#1}],\{(1,(a,b)),(2,(b,c)),(3,(a,c))\},\{(trans,1,2)\},4\rangle && \text{(ACTIVATE)}\\
&\xmapsto[r]{[trans]}{}^{*} \langle [(3,(a,c))^{\#3},(2,(b,c))^{\#1}],\{(1,(a,b)),(2,(b,c)),(3,(a,c))\},\{(trans,1,2)\},4\rangle && (2\times\text{DEFAULT})\\
&\xmapsto[r]{[trans]} \langle [(2,(b,c))^{\#1}],\{(1,(a,b)),(2,(b,c)),(3,(a,c))\},\{(\mathbf{trans},\mathbf{1},\mathbf{2})\},4\rangle && \text{(DROP)}\\
&\xmapsto[r]{[trans]}{}^{*} \langle [(2,(b,c))^{\#3}],\{(1,(a,b)),(2,(b,c)),(3,(a,c))\},\{(trans,1,2)\},4\rangle && (2\times\text{DEFAULT})\\
&\xmapsto[r]{[trans]} \langle [],\{(1,(a,b)),(2,(b,c)),(3,(a,c))\},\{(trans,1,2)\},4\rangle && \text{(DROP)}
\end{aligned}
$$

Figure 1: Demonstration of the effect of the propagation history.

4 FreeCHR

FreeCHR [21] was introduced as a framework to formalize and standardize the embedding of CHR into arbitrary programming languages. The main idea is to model the syntax of programs as a functor within the domain of the host language. We want to briefly reiterate the foundational definitions including the *very abstract* operational semantics. For more details, we refer the reader to the original publication.

Definition 7 (Syntax of FreeCHR programs)**.** The functor

$$\mathrm{CHR}_C D = \mathtt{str} \times \mathtt{list}\, 2^C \times \mathtt{list}\, 2^C \times 2^{\mathtt{list} C} \times (\mathtt{list} C)^{\mathtt{list} C} \sqcup D \times D$$

describes the syntax of FreeCHR programs.

The set $\mathtt{str} \times \mathtt{list}\, 2^C \times \mathtt{list}\, 2^C \times 2^{\mathtt{list} C} \times (\mathtt{list} C)^{\mathtt{list} C}$ is the set of single rules. The name of a rule is a string in `str`. The kept and removed head of a rule are sequences of functions in $\mathtt{list}\, 2^C$ which map elements of C to Booleans, effectively checking individual values for applicability of the rule. The guard of the rule is a function in $2^{\mathtt{list} C}$ and maps sequences of elements in C to Booleans, checking all matched values in the context of each other. Finally, the body of the rule is a function in $(\mathtt{list} C)^{\mathtt{list} C}$ and maps the matched values to a list of newly generated values. The set $D \times D$ represents the composition of FreeCHR programs by an execution strategy, allowing the construction of more complex programs from, ultimately, single rules. By the structure of CHR_C, a CHR_C-algebra with carrier D is defined by two functions

$$\rho : \mathtt{str} \times \mathtt{list}\, 2^C \times \mathtt{list}\, 2^C \times 2^{\mathtt{list} C} \times (\mathtt{list} C)^{\mathtt{list} C} \longrightarrow D \qquad \nu : D \times D \to D$$

$$\frac{S \overset{p_j}{\underset{a\star}{\longmapsto}} S'}{S \overset{p_1 \odot \ldots \odot p_j \odot \ldots \odot p_k}{\underset{a\star}{\longmapsto}} S'} \text{ STEP}$$

$$\frac{k_1(c_1) \wedge \ldots \wedge k_n(c_n) \wedge r_1(c_{n+1}) \wedge \ldots \wedge r_m(c_{n+m}) \wedge g(c_1, \ldots, c_{n+m}) \equiv_2 \mathit{true}}{\{c_1, \ldots, c_{n+m}\} \uplus \Delta S \overset{\mathit{rule}(N,[k_1,\ldots,k_n],[r_1,\ldots,r_m],g,b)}{\underset{a\star}{\longmapsto}} \{c_1, \ldots, c_n\} \uplus b(c_1, \ldots, c_{n+m}) \uplus \Delta S} \text{ APPLY}$$

Figure 2: Inference Rules for the *very abstract* operational semantics of FreeCHR $\omega_a^\star$

as $(D, [\rho, \nu])$. A CHR_C-algebra is called an *instance* of FreeCHR. The free CHR_C-algebra

$$\mathrm{CHR}_C^\star = (\mu\mathrm{CHR}_C, [\mathit{rule}, \odot])$$

with

$$\begin{aligned}\mu\mathrm{CHR}_C = {} & \mathtt{str} \times \mathtt{list}\, 2^C \times \mathtt{list}\, 2^C \times 2^{\mathtt{list}\, C} \times (\mathtt{list}\, C)^{\mathtt{list}\, C} \\ & \sqcup \mu\mathrm{CHR}_C \times \mu\mathrm{CHR}_C\end{aligned}$$

and injections *rule* and $\odot$ provides us with an inductively defined representation of programs. The program from Example 8 (without pattern indices) can be expressed in FreeCHR as shown in Example 10.

Example 10 (Euclidean algorithm (cont.)). The program $gcd = \mathit{zero} \odot \mathit{subtract}$ with

$$\begin{aligned}\mathit{zero} &= \mathit{rule}(\mathtt{zero}, [], [\lambda n.n = 0], (\lambda n.\mathit{true}), (\lambda n.[])) \\ \mathit{subtract} &= \mathit{rule}(\mathtt{subtract}, [\lambda n.0 < n], [\lambda m.0 < m], (\lambda n\ m.n \leq m), (\lambda n\ m.[m-n]))\end{aligned}$$

implements the Euclidean algorithm in FreeCHR. λ-abstractions are used for ad-hoc definitions of functions.

Finally, we want to recall the *very abstract* operational semantics $\omega_a^\star$ of FreeCHR.

Definition 8 (*Very abstract* operational semantics $\omega_a^\star$)**.** The *very abstract* operational semantics of FreeCHR is defined as the labelled transition system

$$\omega_a^\star = \langle\, \mathtt{mset}\, C\ ,\ \mu\mathrm{CHR}_C\ ,\ (\overset{}{\underset{a\star}{\longmapsto}})^*\, \rangle$$

where the transition relation $(\overset{}{\underset{a\star}{\longmapsto}}) \subset \mathtt{mset}\, C \times \mu\mathrm{CHR}_C \times \mathtt{mset}\, C$ is defined by the inference rules described in Figure 2. The functor $\mathtt{mset}$ maps a set X to the set $\mathtt{mset}\, X$ of multisets over X.

Rule selection The STEP transition selects a component program p_j from the composite program $p_1 \odot \ldots \odot p_j \odot \ldots \odot p_k$.

Rule application The APPLY transition applies a rule to the current state of the program if the state contains a unique value for each pattern in the head of the rule and these values satisfy the guard.

$$\frac{c \in C}{\langle c : Q,S,H,I\rangle \xmapsto[r\star]{p} \langle (I,c)^{\#1} : Q,\{(I,c)\} \uplus S,H,I+1\rangle}\ \text{ACTIVATE} \qquad \frac{l \notin indices(p)}{\langle (i,c)^{\#l} : Q,S,H,I\rangle \xmapsto[r\star]{p} \langle Q,S,H,I\rangle}\ \text{DROP}$$

$$\frac{l \in indices(rule(N,k,r,g,b)) \quad \langle (i,c)^{\#l} : Q,S,H,I\rangle \xmapsto[r\star]{rule(N,k,r,g,b)} s}{\langle (i,c)^{\#l} : Q,S,H,I\rangle \xmapsto[r\star]{p_1 \odot ... \odot rule(N,k,r,g,b) \odot ... \odot p_k} s}\ \text{SELECT}$$

$$\frac{(i_a,c_a) \in K \uplus R \quad h_1(c_1) \wedge ... \wedge h_a(c_a) \wedge ... \wedge h_{n+m}(c_{n+m}) \wedge g(c_1,...,c_{n+m}) \equiv_2 true \quad (N,i_1,...,i_{n+m}) \notin H}{\langle (i_a,c_a)^{\#l_a} : Q,K \uplus R \uplus \Delta S,H,I\rangle \xmapsto[r\star]{rule(N,k,r,g,b)} \langle B \diamond (i_a,c_a)^{\#l_a} : Q,K \uplus \Delta S,H \uplus \{(N,i_1,...,i_{n+m})\},I\rangle}\ \text{APPLY}$$

$$\text{with } B = b(c_1,...,c_a,...,c_{n+m}),\ K = \{(i_1,c_1),...,(i_n,c_n)\},\ R = \{(i_{n+1},c_{n+1}),...,(i_{n+m},c_{n+m})\},$$
$$K \uplus R = \{(i_1,c_1),...,(i_a,c_a),...,(i_{n+m},c_{n+m})\} \text{ and } k \diamond r = [h_1^{\#l_1},...,h_a^{\#l_a},...,h_{n+m}^{\#l_{n+m}}]$$

$$\frac{\forall (K \uplus R) \subseteq S.(h_1(c_1) \wedge ... \wedge h_a(c_a) \wedge ... \wedge h_{n+m}(c_{n+m}) \wedge g(c_1,...,c_{n+m}) \equiv_2 false) \vee (N,i_1,...,i_{n+m}) \in H}{\langle (i_a,c_a)^{\#l_a} : Q,S,H,I\rangle \xmapsto[r\star]{rule(N,k,r,g,b)} \langle (i_a,c_a)^{\#l_a+1} : Q,S,H,I\rangle}\ \text{DEFAULT}$$

$$\text{with } K \uplus R = \{(i_1,c_1),...,(i_a,c_a),...,(i_{n+m},c_{n+m})\} \text{ and } k \diamond r = [h_1^{\#l_1},...,h_a^{\#l_a},...,h_{n+m}^{\#l_{n+m}}]$$

Figure 3: Inference Rules for the *refined* operational semantics of FreeCHR $\omega_r^\star$

5 *Refined* operational semantics for FreeCHR

We now introduce the definition of the *refined* operational Semantics $\omega_r^\star$ of FreeCHR.

Definition 9 (*Refined* operational semantics $\omega_r^\star$)**.** The *refined* operational semantics $\omega_r^\star$ for FreeCHR are defined as an LTS

$$\omega_r^\star = \langle\ \Omega_r C\ ,\ \mu\text{CHR}_C^\#\ ,\ (\xmapsto[r\star]{})^+\ \rangle$$

where $(\xmapsto[r\star]{}) \subset \Omega_r C \times \mu\text{CHR}_C^\# \times \Omega_r C$ is defined by the transitions described in Figure 3.

Analogously to $\mathbf{PRG}_C^\#$, $\mu\text{CHR}_C^\#$ is the set of FreeCHR programs in μCHR_C with enumerated patterns (incrementing from right to left and top to bottom). The function *indices* maps programs $p \in \mu\text{CHR}_C^\#$ to the set of pattern indices occurring in p.

Activate The ACTIVATE transition is a direct translation of the ACTIVATE rule of ω_r. It introduces the value together with a unique identifier to the store and decorates the value on the query with that identifier as well as an initial pattern index.

Drop The DROP transition is the direct translations of the DROP rule of ω_r. It removes the currently active value from the query if its pattern index exceeds the indices of the program.

Select The SELECT transition initiates the APPLY or DEFAULT transitions by selecting the rule $rule(N,k,r,g,b)$ from the composition $p_1 \odot ... \odot rule(N,k,r,g,b) \odot ... \odot p_k$ which contains the pattern index of the currently active value.

Apply The APPLY transition, is a translation of the APPLY transition of ω_r. It applies the rule $rule(N,k,r,g,b)$ to the state if c_a satisfies the pattern h_a (which is determined by the label index l_a), the store contains a pair (i_ι, c_ι) for every other pattern h_ι, such that $h_\iota(c_\iota)$ evaluates to *true*, the guard $g(c_1,...,c_{n+m})$ evaluates to *true*, and the configuration $(N,i_1,...,i_{n+m})$ is not already recorded in the propagation history. If all conditions are met, the values $\{(i_{n+1},c_{n+1}),...,(i_{n+m},c_{n+m})\}$ are removed from the store, the values B generated by the body $b(c_1,...,c_{n+m})$ are queried and the configuration $(N,i_1,...,i_{n+m})$ is recorded in the propagation history, to prevent reapplication.

Default Finally, the transition is a translation of the DEFAULT rule of ω_r. It can be applied if no other transition is applicable.

The *refined* semantics $\omega_r^\star$ for FreeCHR are mostly a direct translation of the *refined* semantics for CHR and operate on the same kind of states. The only major difference is that the transition rule STEP provides a proxy for APPLY and DEFAULT. STEP demands to select a rule which has a pattern with the pattern index of the currently active value.

6 Correctness w.r.t. the very abstract operational semantics of FreeCHR

We now want to establish the *refined* semantics $\omega_r^\star$ as a valid concretization of the *very abstract* semantics $\omega_a^\star$. The full proofs of this section and of Section 7 can be found in the preprint[24].

Since the semantics are defined on different domains of states, we first define a function which strips the additional decoration needed by the *refined* semantics.

Definition 10 (Abstraction function)**.** The function

$$abstract_r : \Omega_r C \longrightarrow \texttt{mset}\, C$$
$$abstract_r\langle Q,S,_,_\rangle = \{c \mid (_,c) \in S\} \uplus \{c \mid c \in Q \cap C\}$$

transforms the states of $\Omega_r C$ into multisets in $\texttt{mset}\, C$.

We use a similar approach to the proof of soundness of ω_r w.r.t. the *theoretical* operational semantics ω_t [9]. Additionally, the function *unenum* removes all pattern indices from a program.

Theorem 1. *The* refined *operational semantics of FreeCHR* $\omega_r^\star$ *is* $(abstract_r, unenum)$*-sound w.r.t. the* very abstract *operational semantics* $\omega_a^\star$.

Proof sketch. We show that

$$abstract_r(s) \xmapsto[a\star]{unenum(p)} abstract_r(s') \in (\underset{a\star}{\longmapsto})^* \text{ if } s \xmapsto[r\star]{p} s' \in (\underset{r\star}{\longmapsto})$$

via induction over the inference rules of $(\underset{r\star}{\longmapsto})$. Theorem 1 then follows from the properties of the transitive closure. The first base cases ACTIVATE, DROP and DEFAULT can be reduced to reflexive transitions, since the modifications performed by them are discarded by $abstract_r$. The APPLY case can be reduced to the APPLY rule of $\omega_a^\star$. Finally, the induction step SELECT can be reduced to the STEP transition of $\omega_a^\star$.

q.e.d.

With Theorem 1, we established the *refined* semantics $\omega_r^\star$ as a valid concretization of $\omega_a^\star$. Since $\omega_a^\star$ is already proven to be sound and complete w.r.t. ω_a, Theorem 1 already establishes possible implementations of $\omega_r^\star$ as correct implementations of CHR. To show that such implementations also behave like existing implementations of CHR, we will continue to prove soundness and completeness of $\omega_r^\star$ w.r.t. the *refined* semantics ω_r of CHR.

7 Correctness w.r.t. the refined operational semantics of CHR

In order to prove correctness of $\omega_r^\star$ w.r.t. ω_r, we first need to embed FreeCHR programs in $\text{CHR}_C^\#$ into classical CHR programs in $\mathbf{PRG}_C^\#$.

Definition 11 (Embedding FreeCHR into CHR)**.** The embedding function $embed^\#$ embeds enumerated FreeCHR programs into enumerated CHR programs in $\mathbf{PRG}_C^\#$. It is defined as

$$
\begin{aligned}
&embed^\# : \mu\text{CHR}_C^\# \longrightarrow \mathbf{PRG}_C^\# \\
&embed^\#(rule(N, [h_1^{\#l_1}, ..., h_n^{\#l_n}], [h_{n+1}^{\#l_{n+1}}, ..., h_{n+m}^{\#l_{n+m}}], g, b)) \\
&\qquad = [N @ c_1^{\#l_1}, ..., c_n^{\#l_n} \setminus c_{n+1}^{\#l_{n+1}}, ..., c_{n+m}^{\#l_{n+m}} \\
&\qquad\quad \Leftrightarrow h_1(c_1) \wedge ... \wedge h_{n+m}(c_{n+m}) \wedge g(c_1, ..., c_{n+m}) \mid b(c_1, ..., c_{n+m})] \\
&embed^\#(p_1 \odot ... \odot p_l) = embed^\#(p_1) \diamond ... \diamond embed^\#(p_l)
\end{aligned}
$$

The embedding works by checking the pattern predicates and the guard of the FreeCHR rule in the guard of the CHR rule.

Example 11 (Embedding FreeCHR into CHR). If we apply the embedding $embed^\#$ to the enumerated version of the program *gcd* (including the evaluation of function applications), we get the program

$$
\begin{aligned}
zero \ @ \ n^{\#1} &\ \Leftrightarrow \ n = 0 \wedge true \mid \emptyset \\
subtract \ @ \ n^{\#3} \setminus m^{\#2} &\ \Leftrightarrow \ 0 < n \wedge 0 < m \wedge n \leq m \mid m - n
\end{aligned}
$$

with universally quantified variables n and m. The program corresponds to the *head-normalization* [8] of the program in Example 10.

Theorem 2. *The* refined *operational semantics $\omega_r^\star$ of FreeCHR is $(\mathbf{id}_{\Omega_r C}, embed^\#)$-sound w.r.t. the* refined *operational semantics ω_r of CHR.*

Proof sketch. We show that

$$
S \xmapsto[r]{embed^\#(p)} S' \in (\underset{r}{\mapsto})^+ \text{ if } S \xmapsto[r\star]{p} S' \in (\underset{r\star}{\mapsto})
$$

by induction over the inference rules defining $(\underset{r\star}{\mapsto})$. Theorem 2 then follows from the definition of the transitive closure. The first induction base cases ACTIVATE and DROP can be trivially proven by their counterpart. For the APPLY transition, we know that there is a C-instance of the embedded rule that meets the conditions of the APPLY transition of ω_r since we know from the premise that the rule fires. For the DEFAULT transition, we show that, if the premises hold, no other transition of the original definition than DEFAULT can be applied. Finally, for the induction step, we derive from the induction hypothesis that we can perform the transition with $embed^\#(p)$ if we can perform it with $embed^\#(p_j)$, as we are forced to choose p_j anyway. *q.e.d.*

Theorem 2 establishes $\omega_r^\star$ as a valid concretization of ω_r. We continue with stating and proving the opposite direction.

Theorem 3. *The* refined *operational semantics $\omega_r^\star$ of FreeCHR is $(\mathbf{id}_{\Omega_r C}, embed^\#)$-complete w.r.t. the* refined *operational semantics ω_r of CHR.*

Proof sketch. We show that

$$S \underset{r\star}{\overset{p}{\longmapsto}} S' \in \left(\underset{r\star}{\longmapsto}\right)^+ \text{ if } \underset{r}{\overset{embed^\#(p)}{\longmapsto}} S' \in \left(\underset{r}{\mapsto}\right)$$

from which Theorem 3 follows by the properties of the transitive clause. The ACTIVATE and DROP transitions are again trivial to verify. If we can apply the APPLY transition for an embedded rule, we know that we can use the SELECT transition to select the original rule. Since there is a C-instance for the embedded rule, we also know that APPLY transition of $\omega_r^\star$ is applicable to the original rule. That the conditions are met is derived from the assumed premises. Finally, the DEFAULT transition is verified by assuming the premise that there is no C-instance of the embedded rule we currently need to apply. With this we use the SELECT transition to select the rule and then the DEFAULT transition. That the condition of the DEFAULT transition is met follows from the premise. *q.e.d.*

With Theorem 2 and Theorem 3 we established $\omega_r^\star$ and ω_r as equivalent representations of each other.

8 Conclusion

In this paper we presented a definition of the *refined* semantics $\omega_r^\star$ of FreeCHR. We proved soundness of our definition w.r.t. the *very abstract* semantics $\omega_a^\star$ of FreeCHR and thereby established $\omega_r^\star$ as a valid concretization of $\omega_a^\star$. We also proved soundness and completeness of $\omega_r^\star$ w.r.t. the original *refined* semantics ω_r of CHR and thereby establishes $\omega_r^\star$ and ω_r as valid representations of each other and hence a FreeCHR implementation with $\omega_r^\star$ semantics as a correct implementation of CHR.

This provides the formal foundation for implementations of FreeCHR which are as expressive and behave like the established implementations of CHR. Defining a host language agnostic *algorithmic* representation of the *refined* semantics is the subject of ongoing work [22, 23]. It will serve as a blueprint to implement consistent and correct CHR systems and also a platform for the aggregation of existing and new improvements which can then be applied consistently to existing implementations. Example implementations of FreeCHR with refined semantics can be found on *GitLab*[3].

Future work will mostly be concerned with improvements by adapting and applying known optimization techniques for CHR (see, e.g. [29]). We plan to accompany the optimizations both with benchmarks to validate their effectiveness and proofs of correctness to ensure the formal validity of our approach. This is especially important, as optimizations are often accompanied by deviation from the formal definition. This causes a gap between formalism and programming language which FreeCHR is designed to close.

Acknowledgements

We thank the anonymous reviewers for their constructive and mindful feedback.

[3] `https://gitlab.com/freechr/`

References

[1] Slim Abdennadher, Thom Frühwirth & Holger Meuss (1996): *On Confluence of Constraint Handling Rules*. In Eugene C. Freuder, editor: *Principles and Practice of Constraint Programming — CP96*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 1–15, doi:10.1007/3-540-61551-2_62.

[2] Slim Abdennadher, Ekkerhard Krämer, Matthias Saft & Matthias Schmauss (2002): *JACK: A Java Constraint Kit*. In: *Electronic Notes in Theoretical Computer Science*, 64, pp. 1–17, doi:10.1016/S1571-0661(04)80344-X.

[3] Slim Abdennadher & Michael Marte (2000): *University Course Timetabling Using Constraint Handling Rules*. *AAI* 14(4), pp. 311–325, doi:10.1080/088395100117016.

[4] Vincent Barichard (2024): *CHR++: An Efficient CHR System in C++ with Don't Know Non-Determinism*. *Expert Systems with Applications* 238, p. 121810, doi:10.1016/j.eswa.2023.121810.

[5] W. Chin, M. Sulzmann & Meng Wang (2008): *A Type-Safe Embedding of Constraint Handling Rules into Haskell*. Available at `https://www.semanticscholar.org/paper/A-Type-Safe-Embedding-of-Constraint-Handling-Rules-Chin-Sulzmann/ea47790fc268710d73b2a6be0305e3f3453682e3`.

[6] Henning Christiansen & Maja H. Kirkeby (2015): *Confluence Modulo Equivalence in Constraint Handling Rules*. In Maurizio Proietti & Hirohisa Seki, editors: *Logic-Based Program Synthesis and Transformation*, Lecture Notes in Computer Science, Springer International Publishing, Cham, pp. 41–58, doi:10.1007/978-3-319-17822-6_3.

[7] Leslie De Koninck, Tom Schrijvers, Bart Demoen, M. Fink, H. Tompits & S. Woltran (2006): *INCLP(R) - Interval-based Nonlinear Constraint Logic Programming over the Reals*. In: *WLP 2006*, 1843-06-02, Technische Universität Wien, Austria. Available at `https://lirias.kuleuven.be/1653773`.

[8] Gregory J Duck (2005): *Compilation of Constraint Handling Rules*. Ph.D. thesis, University of Melbourne, Victoria, Australia.

[9] Gregory J. Duck, Peter J. Stuckey, María García de la Banda & Christian Holzbaur (2004): *The Refined Operational Semantics of Constraint Handling Rules*. In Bart Demoen & Vladimir Lifschitz, editors: *ICLP 2004*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 90–104, doi:10.1007/978-3-540-27775-0_7.

[10] Thom Fruehwirth (2006): *Complete Propagation Rules for Lexicographic Order Constraints over Arbitrary Domains*. In Brahim Hnich, Mats Carlsson, François Fages & Francesca Rossi, editors: *CSCLP 2005*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 14–28, doi:10.1007/11754602_2.

[11] Thom Fruehwirth (2009): *Constraint Handling Rules*. Cambridge University Press, Cambridge, U.K. New York.

[12] Thom Fruehwirth (2015): *Constraint Handling Rules - What Else?* In Nick Bassiliades, Georg Gottlob, Fariba Sadri, Adrian Paschke & Dumitru Roman, editors: *RuleML 2015*, Lecture Notes in Computer Science, Springer International Publishing, Cham, pp. 13–34, doi:10.1007/978-3-319-21542-6_2.

[13] T. Frühwirth (2025): *Principles of Rule-Based Programming*. BoD - Books on Demand. Available at `https://books.google.de/books?id=GlNQEQAAQBAJ`.

[14] Daniel Gall & Thom Frühwirth (2017): *A Decidable Confluence Test for Cognitive Models in ACT-R*. In Stefania Costantini, Enrico Franconi, William Van Woensel, Roman Kontchakov, Fariba Sadri & Dumitru Roman, editors: *Rules and Reasoning*, Springer International Publishing, Cham, pp. 119–134, doi:10.1007/978-3-319-61252-2_9.

[15] P. Hudak (1998): *Modular Domain Specific Languages and Tools*. In: *ICSR 1998*, pp. 134–142, doi:10.1109/ICSR.1998.685738.

[16] Dragan Ivanović (2013): *Implementing Constraint Handling Rules as a Domain-Specific Language Embedded in Java*, doi:10.48550/arXiv.1308.3939. arXiv:1308.3939.

[17] Patricia Johann & Neil Ghani (2007): *Initial Algebra Semantics Is Enough!* In Simona Ronchi Della Rocca, editor: *TLCA 2007*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 207–222, doi:10.1007/978-3-540-73228-0_16.

[18] E. Lam & Martin Sulzmann (2006): *Towards Agent Programming in CHR*. Available at `https://www.semanticscholar.org/paper/Towards-Agent-Programming-in-CHR-Lam-Sulzmann/43277216bafb824d651c802ad487dbce8f7f7478`.

[19] Edmund S. L. Lam & Martin Sulzmann (2007): *A Concurrent Constraint Handling Rules Implementation in Haskell with Software Transactional Memory*. In: *DAMP '07*, ACM Press, Nice, France, pp. 19–24, doi:10.1145/1248648.1248653.

[20] Falco Nogatz, Thom Frühwirth & Dietmar Seipel (2018): *CHR.Js: A CHR Implementation in JavaScript*. In Christoph Benzmüller, Francesco Ricca, Xavier Parent & Dumitru Roman, editors: *RuleML 2018*, Lecture Notes in Computer Science, Springer International Publishing, Cham, pp. 131–146, doi:10.1007/978-3-319-99906-7_9.

[21] Sascha Rechenberger & Thom Frühwirth (2025): *FreeCHR – An Algebraic Framework for Constraint Handling Rules Embeddings*. *Theory and Practice of Logic Programming*, pp. 1–34, doi:10.1017/S1471068425000043.

[22] Sascha Rechenberger & Thom Frühwirth (2025): *An Instance of FreeCHR with Refined Operational Semantics*, doi:10.48550/arXiv.2505.22155. arXiv:2505.22155.

[23] Sascha Rechenberger & Thom Frühwirth (2025): *Optimized Execution of FreeCHR*, doi:10.48550/arXiv.2506.14485. arXiv:2506.14485.

[24] Sascha Rechenberger & Thom Frühwirth (2025): *A Refined Operational Semantics for FreeCHR*, doi:10.48550/arXiv.2504.04962. arXiv:2504.04962.

[25] Tom Schrijvers & Bart Demoen (2004): *The K.U. Leuven CHR System: Implementation and Application*. In: *CHR 2004*, pp. 1–5. Available at `https://lirias.kuleuven.be/retrieve/33588`.

[26] Jon Sneyers, Peter Van Weert, Tom Schrijvers & Leslie De Koninck (2010): *As Time Goes by: Constraint Handling Rules: A Survey of CHR Research from 1998 to 2007*. *TPLP* 10(1), pp. 1–47, doi:10.1017/S1471068409990123.

[27] Michael Thielscher (2002): *Reasoning about Actions with CHRs and Finite Domain Constraints*. In Peter J. Stuckey, editor: *LP 2002*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 70–84, doi:10.1007/3-540-45619-8_6.

[28] Michael Thielscher (2005): *FLUX: A Logic Programming Method for Reasoning Agents*. *TPLP* 5(4-5), pp. 533–565, doi:10.1017/S1471068405002358.

[29] Peter van Weert (2010): *Efficient Lazy Evaluation of Rule-Based Programs*. *IEEE Trans. Knowl. Data Eng.* 22(11), pp. 1521–1534, doi:10.1109/tkde.2009.208.

[30] Peter Van Weert, Tom Schrijvers, Bart Demoen, Tom Schrijvers & Thom Frühwirth (2005): *K.U.Leuven JCHR: A User-Friendly, Flexible and Efficient CHR System for Java*. In: *CHR 2005*, Deptartment of Computer Science, K.U.Leuven. Available at `https://lirias.kuleuven.be/1654703`.

[31] Tim Wibiral (2022): *JavaCHR – A Modern CHR-Embedding in Java*. Bachelor thesis, Universität Ulm. Available at `https://oparu.uni-ulm.de/xmlui/handle/123456789/43506`.

[32] Pieter Wuille, Tom Schrijvers & Bart Demoen (2007): *CCHR: The Fastest CHR Implementation, in C*. In: *CHR 2007*, pp. 123–137. Available at `https://lirias.kuleuven.be/retrieve/22123`.

On the Trap Space Semantics of Normal Logic Programs

Van-Giang Trinh Sylvain Soliman François Fages
Inria Saclay, EP Lifeware, Palaiseau, France
van-giang.trinh@inria.fr Sylvain.Soliman@inria.fr francois.fages@inria.fr
Belaid Benhamou
LIRICA team, LIS, Aix-Marseille University, Marseille, France
belaid.benhamou@lis-lab.fr

The logical semantics of normal logic programs has traditionally been based on the notions of Clark's completion and two-valued or three-valued canonical models, including supported, stable, regular, and well-founded models. Two-valued interpretations can also be seen as states evolving under a program's update operator, producing a transition graph whose fixed points and cycles capture stable and oscillatory behaviors, respectively. We refer to this view as *dynamical semantics* since it characterizes the program's meaning in terms of state-space trajectories, as first introduced in the stable (supported) class semantics. Recently, we have established a formal connection between Datalog$^\neg$ programs (i.e., normal logic programs without function symbols) and Boolean networks, leading to the introduction of the *trap space* concept for Datalog$^\neg$ programs. In this paper, we generalize the trap space concept to arbitrary normal logic programs, introducing *trap space semantics* as a new approach to their interpretation. This new semantics admits both model-theoretic and dynamical characterizations, providing a comprehensive approach to understanding program behavior. We establish the foundational properties of the trap space semantics and systematically relate it to the established model-theoretic semantics, including the stable (supported), stable (supported) partial, regular, and L-stable model semantics, as well as to the dynamical stable (supported) class semantics. Our results demonstrate that the trap space semantics offers a unified and precise framework for proving the existence of supported classes, strict stable (supported) classes, and regular models, in addition to uncovering and formalizing deeper relationships among the existing semantics of normal logic programs.

Keywords: logic programming, normal logic program, Clark's completion, model-theoretic semantics, dynamical semantics, trap space, Boolean network, infinite structure

1 Introduction

The meaning of a Normal Logic Program (NLP)—its declarative semantics—is typically defined by a class of models of a logical formula associated with the program, such as the Herbrand models of the *Clark's completion* of the program [13]. This was later shown to be equivalent to the *supported model semantics* [2]. For NLPs with negation, the Clark's completion may have no model, or several minimal models; therefore, there is generally no unique least Herbrand model. In this setting, the *stable model semantics* was introduced by [24] to provide a canonical notion of two-valued models, defined by an elegant fixed-point definition based on the least model of a *reduct* transformation. Stable models were later characterized as *well-supported models* [21], i.e., models with circuit-free finite justifications—which led to graphical conditions for the existence of stable models [20, 1], and equivalence results with other forms of non-monotonic logics [43, 31, 32, 7].

A three-valued logic approach was introduced, where the third value denotes ignorance [23, 33, 35]. Subsequent research expanded the semantic landscape of NLPs through additional three-valued logic

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 294–319, doi:10.4204/EPTCS.439.21

© V.G. Trinh, S. Soliman, F. Fages & B. Benhamou
This work is licensed under the
Creative Commons Attribution License.

transformations [3]. The *stable (supported) partial model semantics*—also known as the three-valued stable (supported) model semantics—extends the two-valued stable (supported) model semantics [34, 19, 33]. Under this framework, all models are minimal, and every NLP possesses at least one stable (supported) partial model. Notably, two specific types of stable partial semantics are distinguished by their informational content: the *well-founded partial model semantics* [22, 33], representing the unique model with minimal information, and the *regular model semantics* [42], encompassing models with maximal information. It is worth noting that these three-valued semantics (e.g., stable partial models, regular models) in normal logic programs have been shown to correspond to several extension-based semantics (e.g., complete extensions, preferred extensions, respectively) in Dung's frameworks [17, 41] and assumption-based argumentation [10], which are two prominent formalisms in symbolic AI.

Alternatively, an NLP can be viewed as a state-transition system in which each rule in the program represents how a component (here an atom) is affected by other components [4, 8]. The dynamical behaviors of the program, including its oscillating behavior [27], are characterized by a directed graph called the (stable or supported) *transition graph* whose vertices are two-valued interpretations and whose arcs are transitions between interpretations. This led to the *stable class semantics* [4] and *supported class semantics* [27]. If a stable (resp. supported) class is of size one, it coincides with a stable (resp. supported) model and exhibits a stable behavior. Otherwise, it exhibits an oscillating behavior.

The aforementioned semantics capture various aspects of program behavior. Despite their success, they often diverge in interpretive principles, and many lack a unifying framework that integrates both model-theoretic and dynamical behaviors coherently. Recently, we have established a formal connection between Datalog$^\neg$ programs (i.e., NLPs without function symbols—thus the Herbrand base is finite) and Boolean networks, leading to the introduction of the concept of *trap space* for Datalog$^\neg$ programs, inspired by the notion of trap space in Boolean networks [28, 39]. This concept offers a bridge between model-theoretic and dynamical aspects, but applies only to finite structures [40].

In this paper, we generalize the trap space concept to arbitrary NLPs, introducing *trap space semantics* as a new approach to their interpretation. This new semantics admits both model-theoretic and dynamical characterizations, providing a comprehensive approach to understanding program behavior. We establish foundational properties of the trap space semantics and systematically relate it to established model-theoretic and dynamical semantics—including the stable (supported), stable (supported) partial, regular, and L-stable model semantics—as well as to dynamical semantics such as the stable (supported) class semantics. Our results demonstrate that the trap space semantics offers a unified framework (along with a proper use of set-theoretic axioms) for proving the existence of supported classes, strict supported classes, strict stable classes, or regular models—proofs that have been unfortunately either missing or imprecise in prior literature [4, 42, 27]. Furthermore, by leveraging the concept of stable or supported trap set underlying the trap space semantics, we correct a flaw in the existing alternative characterization of strict stable or supported classes established in [27]. Finally, the trap space semantics emerges as a powerful tool for uncovering and formalizing deeper relationships among various existing semantics of NLPs, for instance, the stable (resp. supported) class semantics vs. the stable (resp. supported) partial model semantics, and the regular model semantics vs. the stable class semantics and the L-stable model semantics.

The rest of the paper is organized as follows: in Section 2, we review the necessary background on NLPs and set theory. Section 3 presents the formal definitions of the trap space semantics, as well as its fundamental properties. Section 4 shows the relationships between the trap space semantics and other semantics. In Section 5, we present the existence results. Section 6 offers concluding remarks and future research directions.

2 Preliminaries

2.1 Normal Logic Programs

We assume that the reader is familiar with the logic programming theory and stable model semantics. Unless specifically stated, NLP means normal logic program. In addition, we consider the Boolean domain $\mathbb{B} = \{0,1\}$, the three-valued domain $\mathbb{B}_\star = \{0,1,\star\}$. The logical connectives used in this paper are $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), and $\leftrightarrow$ (equivalence).

We consider a first-order language built over an infinite alphabet of variables, and finite alphabets of constant, function, and predicate symbols. The set of first-order *terms* is the least set containing variables and constants, and closed by application of function symbols. An *atom* is a formula of the form $p(t_1,\ldots,t_k)$ where p is a predicate symbol and each t_i is a term. A *normal logic program* P is a *finite* set of *rules* of the form $p \leftarrow p_1,\ldots,p_m,\sim p_{m+1},\ldots,\sim p_k$ where p and p_i are atoms, $k \geq m \geq 0$, and $\sim$ denotes default negation. A fact is a rule with $k = 0$. For any rule r of the above form, $\mathsf{h}(r) = p$ is called the *head* of r, $\mathsf{b}^+(r) = \{p_1,\ldots,p_m\}$ is called the *positive body* of r, $\mathsf{b}^-(r) = \{p_{m+1},\ldots,p_k\}$ is called the *negative body* of r, and $\mathsf{b}(r) = \mathsf{b}^+(r) \cup \mathsf{b}^-(r)$ is called the *body* of r. For convenience, we denote $\mathsf{bf}(r) = \bigwedge_{v\in\mathsf{b}^+(r)} v \wedge \bigwedge_{v\in\mathsf{b}^-(r)} \neg v$ as the *body formula* of r; if $k = 0$, then $\mathsf{bf}(r) = 1$. If $\mathsf{b}^-(r) = \emptyset$ forall $r \in P$, then P is called a *positive* NLP. If $\mathsf{b}^+(r) = \emptyset$ forall $r \in P$, then P is called a *negative* NLP. The program P is called *internal-variable-free* if for any rule in P, the variables in the body also appear in the head [36].

A term, an atom, or an NLP is *ground* if it contains no variables. The *Herbrand base* of an NLP P (denoted by HB_P) is the set of ground atoms formed over the alphabet of P. It is finite in the absence of function symbols, which is the case of *Datalog*$^\neg$ programs [12]. The *ground instantiation* of an NLP P (denoted by $\mathsf{gr}(P)$) is the set of the ground instances of all rules in P. An NLP P is called *uni-rule* if every atom in HB_P appears in the head of at most one rule in $\mathsf{gr}(P)$ [37].

We shall use the *least fixpoint* transformation of NLPs [18] to prove many new results in the next sections. To be self-contained, we briefly recall the definition of the *least fixpoint* of an NLP P as follows. The reader can refer [18] for more details. Let r be the rule $p \leftarrow \sim p_1,\ldots,\sim p_k,q_1,\ldots,q_j$ in $\mathsf{gr}(P)$ and let r_i be the rules $q_i \leftarrow \sim q_i^1,\ldots,\sim q_i^{l_i}$ in $\mathsf{gr}(P)$ where $1 \leq i \leq j$ and $l_i \geq 0$. Then $\Pi_r(\{r_1,\ldots,r_j\})$ is the following rule

$$p \leftarrow \sim p_1,\ldots,\sim p_k,\sim q_1^1,\ldots,\sim q_1^{l_1},\ldots,\sim q_j^1,\ldots,\sim q_j^{l_j},$$

which means that each atom q_j in the positive body of r is substituted with the body of the rule r_j. Then Π_P is the transformation on negative normal logic programs:

$$\Pi_P(Q) = \{\Pi_r(\{r_1,\ldots,r_j\}) \mid r \in \mathsf{gr}(P), r_i \in Q, 1 \leq i \leq j\}.$$

Let $\mathsf{lfp}(P)_i = (\Pi_P(\emptyset))^i = \Pi_P(\Pi_P(\ldots\Pi_P(\emptyset)))$, then $\mathsf{lfp}(P) = \bigcup_{i\geq 1}\mathsf{lfp}(P)_i$ is the least fixpoint of the NLP P.

2.1.1 Stable and Supported Partial Models

A *three-valued interpretation* I of an NLP P is a mapping $I\colon \mathsf{HB}_P \to \mathbb{B}_\star$. If $I(a) \neq \star$ for all $a \in \mathsf{HB}_P$, then I is a *two-valued interpretation* of P. Usually, a two-valued interpretation is written as the set of atoms that are true. A three-valued interpretation I characterizes the set of two-valued interpretations denoted by $\mathscr{S}(I)$ as $\mathscr{S}(I) = \{J \mid J \in 2^{\mathsf{HB}_P}, \forall a \in \mathsf{HB}_P, I(a) \neq \star \Rightarrow J(a) = I(a)\}$. For example, if $I = \{p = 1, q = 0, r = \star\}$, then $\mathscr{S}(I) = \{\{p\},\{p,r\}\}$. We consider three orders on three-valued interpretations. The truth

order $\leq_t$ is given by $0 <_t \star <_t 1$. Then, $I_1 \leq_t I_2$ iff $I_1(a) \leq_t I_2(a)$ forall $a \in \mathsf{HB}_P$. The order $\leq_s$ is given by $0 <_s \star$, $1 <_s \star$; no other relations hold. Then, $I_1 \leq_s I_2$ iff $I_1(a) \leq_s I_2(a)$ forall $a \in \mathsf{HB}_P$. In addition, $I_1 \leq_s I_2$ iff $\mathscr{S}(I_1) \subseteq \mathscr{S}(I_2)$, i.e., $\leq_s$ is identical to the subset partial order. The information order $\leq_i$ is given by $\star <_i 0$, $\star <_i 1$; no other relations hold. Then, $I_1 \leq_i I_2$ iff $I_1(a) \leq_i I_2(a)$ forall $a \in \mathsf{HB}_P$. It is easy to see that $\leq_i$ is the complement of $\leq_s$, i.e., $I_1 \leq_s I_2$ iff $I_2 \leq_i I_1$.

Let e be a propositional formula on HB_P, using only the $\neg$, $\wedge$, and $\vee$ connectives. Then the valuation of e under a three-valued interpretation I (denoted by $I(e)$) following the three-valued logic is defined recursively as follows: $I(e) = e$ if $e \in \mathbb{B}_\star$; $I(e) = I(a)$ if $e = a, a \in \mathsf{HB}_P$; $I(e) = \neg I(e_1)$ if $e = \neg e_1$; $I(e) = \min_{\leq_t}(I(e_1), I(e_2))$ if $e = e_1 \wedge e_2$; and $I(e) = \max_{\leq_t}(I(e_1), I(e_2))$ if $e = e_1 \vee e_2$ where $\neg 1 = 0, \neg 0 = 1, \neg\star = \star$, and $\min_{\leq_t}$ (resp. $\max_{\leq_t}$) is the function to get the minimum (resp. maximum) value of two values w.r.t. the order $\leq_t$. We say three-valued interpretation I is a *three-valued model* of an NLP P if for each rule $r \in \mathsf{gr}(P)$, $I(\mathsf{bf}(r)) \leq_t I(\mathsf{h}(r))$.

Given a three-valued interpretation I of an NLP P, the *reduct* of P w.r.t. I (denoted by P^I) is defined as: remove any rule $a \leftarrow a_1, \ldots, a_m, \sim b_1, \ldots, \sim b_k \in \mathsf{gr}(P)$ if $I(b_i) = 1$ for some $1 \leq i \leq k$; afterwards, remove any occurrence of $\sim b_i$ from $\mathsf{gr}(P)$ such that $I(b_i) = 0$; then, replace any occurrence of $\sim b_i$ left by a special atom $\mathbf{u}$ such that $\mathbf{u} \notin \mathsf{HB}_P$ and it always receives the value $\star$. The NLP P^I is positive and has a unique $\leq_t$-least three-valued model. See [33] for the method for computing this $\leq_t$-least model. Then I is a *stable partial model* of P if I is equal to the $\leq_t$-least three-valued model of P^I. A stable partial model I is an *M-stable model* if it is $\leq_s$-minimal [35]. In this paper, we define a *regular model* as an M-stable model as they are equivalent [19]. In the two-valued setting, P^I is identical to the reduct defined by [24]; I is a *stable model* of P if I is equal to the $\leq_t$-least two-valued model of P^I. It is easy to derive that a stable model is a two-valued regular model, as well as a two-valued stable partial model [3]. We define $I^{\mathbf{u}}$ as the set of ground atoms that are assigned to $\star$ in I, i.e., $I^{\mathbf{u}} = \{v \mid v \in \mathsf{HB}_P, I(v) = \star\}$. Then I is a *L-stable model* of P if I is a stable partial model of P and there is no stable partial model J such that $J^{\mathbf{u}} \subset I^{\mathbf{u}}$ [35].

The (propositional) *Clark's completion* of an NLP P (denoted by $\mathsf{comp}(P)$) consists of the following equivalences: $p \leftrightarrow \bigvee_{r \in \mathsf{gr}(P), \mathsf{h}(r) = p} \mathsf{bf}(r)$, for each $p \in \mathsf{HB}_P$; if there is no rule whose head is p, then the equivalence is $p \leftrightarrow 0$. Let $\mathsf{rhs}_P(a)$ denote the right-hand side of the equivalence of a ground atom $a \in \mathsf{HB}_P$ in $\mathsf{comp}(P)$. A three-valued interpretation I is a three-valued model of $\mathsf{comp}(P)$ iff for every $a \in \mathsf{HB}_P$, $I(a) = I(\mathsf{rhs}_P(a))$. In this work, we define a *supported partial model* of P as a three-valued model of $\mathsf{comp}(P)$, and a *supported model* of P as a two-valued model of $\mathsf{comp}(P)$. Clearly, a supported model is a (two-valued) partial supported model. It has been pointed out that a stable partial model (resp. stable model) is a supported partial model (resp. supported model), but the reverse may be not true [14].

2.1.2 Stable and Supported Classes

Let P be an NLP and let I be any two-valued interpretation of P. We have that P^I is positive, and has a unique $\leq_t$-least two-valued model (say J). We define the operator F_P as $F_P(I) = J$. In contrast, we define the operator T_P as $T_P(I) = J$ where J is a two-valued interpretation such that for every $a \in \mathsf{HB}_P$, $J(a) = I(\mathsf{rhs}_P(a))$.

Definition 2.1. *A non-empty set S of two-valued interpretations is a* stable class *of an NLP P iff it holds that $S = \{F_P(I) \mid I \in S\}$ [4]. Similarly, S is a* supported class *of an NLP P iff it holds that $S = \{T_P(I) \mid I \in S\}$ [27]. A stable (resp. supported) class S of P is* strict *iff no proper subset of S is a stable (resp. supported) class of P.*

Example 2.1. *Consider the NLP $P = \{a \leftarrow b; b \leftarrow a\}$ where the symbol ";" is used to separate program rules. Since P is positive, for any two-valued interpretation I, $P^I = P$. It follows that $F_P(I) = \emptyset$ for all I.*

It is easy to derive that P has only a unique stable class $\{\emptyset\}$. *On the other hand, we have that* $T_P(\emptyset) = \emptyset$, $T_P(\{a\}) = \{b\}$, $T_P(\{b\}) = \{a\}$, *and* $T_P(\{a,b\}) = \{a,b\}$. *It is easy to derive that P has three supported classes:* $\{\emptyset\}$, $\{\{a\},\{b\}\}$, *and* $\{\{a,b\}\}$.

2.1.3 Transition Graphs

The *stable* (resp. *supported*) *transition graph* of P is a (possibly infinite) directed graph (denoted by $\mathtt{tg}_{st}(P)$ (resp. $\mathtt{tg}_{sp}(P)$)) on the set of all possible two-valued interpretations of P such that (I,J) is an arc of $\mathtt{tg}_{st}(P)$ (resp. $\mathtt{tg}_{sp}(P)$) iff $J = F_P(I)$ (resp. $J = T_P(I)$). Given a directed graph G, we use $V(G)$ (resp. $E(G)$) to denote the set of vertices (resp. arcs) of G. For more details about theory of finite and infinite graphs, we refer the reader to [29].

Example 2.2. *Consider the NLP* $P = \{p(X) \leftarrow p(s(X))\}$. *The Clark's completion of P includes* $p(0) \leftrightarrow p(s(0))$, $p(s(0)) \leftrightarrow p(s(s(0)))$, ... *It is easy to verify that P has only three supported partial models:* I_1 *where* $I_1(a) = 0$ *for every* $a \in \mathsf{HB}_P$, I_2 *where* $I_2(a) = 1$ *for every* $a \in \mathsf{HB}_P$, *and* I_3 *where* $I_3(a) = \star$ *for every* $a \in \mathsf{HB}_P$. *We have* $P^{I_1} = P^{I_2} = P^{I_3} = P$. *It is easy to derive that* I_1 *is the unique stable partial model of P. It is also the unique regular model, stable model, and L-stable model of P. The stable and supported transition graphs of P are shown in Figure 1 (a) and Figure 1 (b), respectively. The NLP P has only one stable class* $\{I_1\}$, *but three supported classes* $\{I_1\}$, $\{I_2\}$, *and* $\{I_1,I_2\}$. *Then* $\{I_1\}$ *is the unique strict stable class of P, whereas* $\{I_1\}$ *and* $\{I_2\}$ *are strict supported classes of P.*

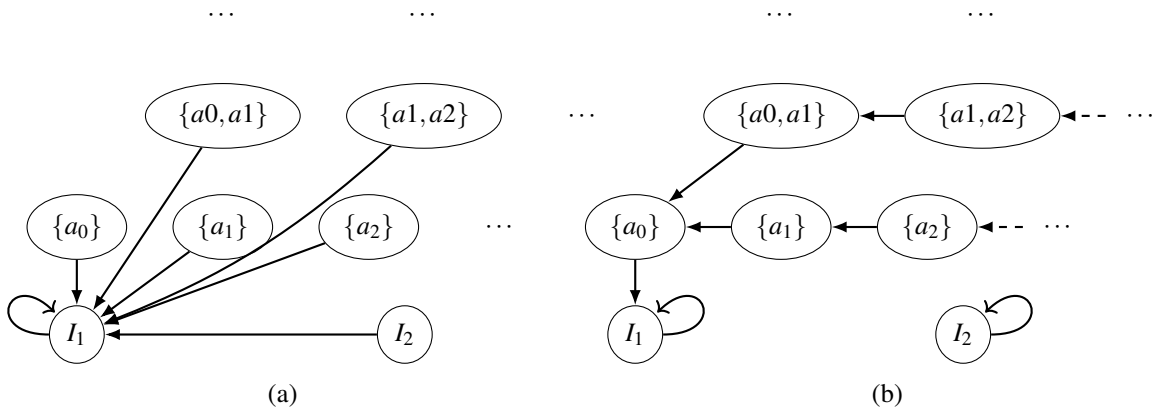


Figure 1: Stable transition graph (a) and supported transition graph (b) of the NLP P of Example 2.2. Herein, a_i denotes $p(s^i(0))$, $I_1(a) = 0, \forall a \in \mathsf{HB}_P$ $(I_1 = \emptyset)$, $I_2(a) = 1, \forall a \in \mathsf{HB}_P$ $(I_2 = \mathsf{HB}_P)$.

2.2 Set Theory

Our proofs in the following sections rely on existing results in set theory. To keep the paper self-contained, we recall them below. For more details, we refer the reader to [25].

Lemma 2.1 (Zorn's lemma [44])**.** *Given a partially ordered set* $(P,\leq)$. *If any* $\leq$*-chain possesses an upper bound, then* $(P,\leq)$ *has a maximal element.*

There is a "strengthened" version that is equivalent to Zorn's lemma:

Lemma 2.2 ([6])**.** *Given a partially ordered set* $(P,\leq)$, *if any* $\leq$*-chain has an upper bound, then for any* $p \in P$, *P has a maximal element m such that* $p \leq m$.

Finally, we show Hartogs' lemma that basically means that for any set X, there is always a bigger ordinal [26].

Lemma 2.3 (Hartogs' lemma)**.** *For any set X, there is a least ordinal α such that there is no injection from α into X.*

3 Trap Space Semantics

The concept of stable (resp. supported) trap set is obtained by relaxing the equality in Definition 2.1 of stable (resp. supported) class, as follows:

Definition 3.1. *A non-empty set S of two-valued interpretations of an NLP P is called a* stable trap set *(resp.* supported trap set*) of P iff $\{F_P(I) \mid I \in S\} \subseteq S$ (resp. $\{T_P(I) \mid I \in S\} \subseteq S$).*

Note that a stable (resp. supported) class is clearly a stable (resp. supported) trap set, but the reverse may not be true. Indeed, given the NLP P of Example 2.2, $\{\emptyset\}$ is a stable class and also a stable trap set, whereas $\{\emptyset, a_0\}$ is a stable trap set but not a stable class. Similarly, $\{\emptyset, \mathsf{HB}_P\}$ is a supported class and also a supported trap set, whereas $\{\emptyset, a_0, a_1\}$ is a supported trap set but not a supported class. By adding the condition on three-valued interpretations, we obtain the concept of stable or supported trap space:

Definition 3.2. *A three-valued interpretation I of an NLP P is called a* stable trap space *(resp.* supported trap space*) of P iff $\mathscr{S}(I)$ is a stable (resp. supported) trap set of P. We call I a* trap space *of P iff it is a stable or supported trap space of P.*

More broadly, it is easy to adapt the concept of stable or supported trap set of an NLP to a (possibly infinite) directed graph in general:

Definition 3.3. *Consider a directed graph G. A non-empty subset S of $V(G)$ is called a* trap set *of G iff there are no two vertices A and B such that $A \in S$, $B \notin S$, and $(A,B) \in E(G)$.*

By definition, this immediately follows that S is a stable (resp. supported) trap set of P iff S is a trap set of $\mathtt{tg}_{st}(P)$ (resp. $\mathtt{tg}_{sp}(P)$). Hence, we can deduce from Definition 3.2 that a three-valued interpretation I is a stable (resp. supported) trap space of P iff $\mathscr{S}(I)$ is a trap set of $\mathtt{tg}_{st}(P)$ (resp. $\mathtt{tg}_{sp}(P)$). Since stable and supported transition graphs represent the dynamical aspect of an NLP [4, 27], this indicates that the trap space semantics for NLPs admits a dynamical characterization.

Example 3.1. *Consider the NLP P of Example 2.2. The stable and supported transition graphs of P are shown in Figure 1 (a) and Figure 1 (b), respectively. We have that the three-valued interpretation I_1 is a stable (resp. supported) trap space of P because $\mathscr{S}(I_1) = \{\emptyset\}$ is a stable (resp. supported) trap set of P. It is easy to verify that $\mathscr{S}(I_1)$ is also a trap set of $\mathtt{tg}_{st}(P)$ (resp. $\mathtt{tg}_{sp}(P)$). Note that $\{\emptyset, a_0, a_1\}$ is a stable (resp. supported) trap set, but it does not correspond to a stable (resp. supported) trap space of P. We see that I_2 is a supported trap space but not a stable trap space of P, since $\mathscr{S}(I_2) = \{\mathsf{HB}_P\}$ is a supported trap set but not a stable trap set of P.*

We define the *trap space semantics* of an NLP as the set of its trap spaces. When applicable we shall restrict ourselves to stable or supported trap spaces to relate those subsets to other semantics. For convenience, let $\mathsf{StTS}(P)$ (resp. $\mathsf{SuTS}(P)$) denote the set of stable (resp. supported) trap spaces of an NLP P. We first establish the simplest property, which is, however, vital for proving subsequent results, especially in infinite structures. Note that, due to space limitations, all proofs are given in Appendix A.

Proposition 3.1. *An NLP P always has a stable or supported trap space.*

Next, we investigate the consistency among stable or supported trap spaces, which shall be used to investigate the model or class disjointness (see Section 4).

Definition 3.4. *Consider an NLP P. Two three-valued interpretations I_1 and I_2 of P are called* consistent *iff for all $a \in \mathsf{HB}_P$, $I_1(a) \leq_s I_2(a)$ or $I_2(a) \leq_s I_1(a)$. Equivalently, I_1 and I_2 are called consistent iff $\mathscr{S}(I_1) \cap \mathscr{S}(I_2) \neq \emptyset$.*

Note that using $I_1 \leq_s I_2$ or $I_2 \leq_s I_1$ in Definition 3.4 is insufficient here, since two three-valued interpretations that are not comparable w.r.t. $\leq_s$ can still be consistent. We then generalize the above definition to a non-empty set of mutually consistent trap spaces:

Definition 3.5. *Given an NLP P, let E be a non-empty set of mutually consistent three-valued interpretations (i.e., E_i and E_j are consistent for all $E_i, E_j \in E$). Then the* intersection *of all elements in E (denoted by $\sqcap_E$) is a three-valued interpretation defined as: $\sqcap_E(a) = min_{\leq_s}\{E_i(a) \mid E_i \in E\}$ for all atom $a \in \mathsf{HB}_P$. This is well-defined because $\{E_i(a) \mid E_i \in E\}$ is a totally ordered finite set w.r.t. $\leq_s$ (thus its minimum element exists), since all elements in E are mutually consistent.*

By pointing out that the intersection of mutually consistent stable (resp. supported) trap spaces is a stable (resp. supported) trap space, along with applying Definition 3.4 and Lemma 2.2, we obtain the following two lemmas:

Lemma 3.1. *Consider an NLP P and a three-valued interpretation J of P. If J is a stable trap space of P, then there exists a $\leq_s$-minimal stable trap space J' of P such that $J' \leq_s J$.*

Lemma 3.2. *Consider an NLP P and a three-valued interpretation J of P. If J is a supported trap space of P, then there exists a $\leq_s$-minimal supported trap space J' of P such that $J' \leq_s J$.*

In addition to Proposition 3.1, we obtain the following main result:

Theorem 3.1 (main result). *Any NLP P has a $\leq_s$-minimal stable (or supported) trap space.*

We then establish the relationship between a non-empty set of two-valued interpretations and stable or supported trap spaces, which is useful for bridging the relationships between stable (resp. supported) classes and stable (resp. supported) trap spaces later.

Proposition 3.2. *Consider an NLP P. Let S be a non-empty set of two-valued interpretations of P. Then there is a unique $\leq_s$-minimal stable (resp. supported) trap space, denoted by $\langle S\rangle_P^{st}$ (resp. $\langle S\rangle_P^{sp}$), covering S, i.e., $S \subseteq \mathscr{S}(\langle S\rangle_P^{st})$ (resp. $S \subseteq \mathscr{S}(\langle S\rangle_P^{sp})$).*

Finally, we show the separation of trap spaces w.r.t. the $\leq_s$-minimality:

Proposition 3.3. *Consider an NLP P. Let I_1 and I_2 be two distinct $\leq_s$-minimal stable (resp. supported) trap spaces of P. Then I_1 and I_2 are not consistent.*

4 Relationships with Other Semantics

Naturally, if a stable (resp. supported) trap space of an NLP P is two-valued, then it is also a stable (resp. supported) model of P. We now present additional relationships between stable or supported trap spaces and other types of models. These results show that the trap space semantics can provide a *unified* semantic framework for NLPs. Note that there are historically different model-theoretic or dynamical semantics for NLPs, and they are often defined in different ways [43, 35]. For convenience, let $\mathsf{StPM}(P)$ (resp. $\mathsf{SuPM}(P)$) denote the set of stable (resp. supported) partial models of P; let $\mathsf{StM}(P)$ (resp. $\mathsf{SuM}(P)$) denote the set of stable (resp. supported) models of P; and let $\mathsf{RegM}(P)$ denote the set of regular models of P.

4.1 Relationships with the Stable and Supported Class Semantics

By leveraging the concept of stable or supported trap set underlying the trap space semantics, we correct a flaw in the existing alternative characterization of strict stable or supported classes proved in [27]. Due to space limitations, we refer the reader to Section B for the technical details.

We now prove a similar version of Lemma 3.1, but for stable trap sets instead of stable trap spaces. The proof technique is analogous to that of Lemma 3.1.

Lemma 4.1. *Consider an NLP P and a non-empty set S of two-valued interpretations of P. If S is a stable trap set of P, then there exists a $\subseteq$-minimal stable trap set S' of P such that $S' \subseteq S$.*

This is similar for supported trap sets:

Lemma 4.2. *Given an NLP P, if a non-empty set S of two-valued interpretations of P is a supported trap set of P, then there exists a $\subseteq$-minimal supported trap set S' of P such that $S' \subseteq S$.*

Based on Lemma 4.1 , we obtain the following main result:

Theorem 4.1 (main result). *Consider an NLP P. A non-empty set S of two-valued interpretations is a $\subseteq$-minimal stable trap set of P iff S is a strict stable class of P.*

Analogously, based on Lemma 4.2 , we obtain the following:

Theorem 4.2. *Consider an NLP P. A non-empty set S of two-valued interpretations is a $\subseteq$-minimal supported trap set of P iff S is a strict supported class of P.*

Theorem 4.1 and Theorem 4.2 immediately entail that every stable (resp. supported) trap space always covers at least one strict stable (resp. supported) class in an NLP:

Corollary 4.1. *Consider an NLP P. For every stable (resp. supported) trap space I of P, $\mathscr{S}(I)$ contains at least one strict stable (resp. supported) class of P.*

They also naturally entail the disjointness of two strict stable or supported classes:

Proposition 4.1. *Two distinct strict stable (resp. supported) classes of an NLP P are disjoint.*

In [4], the authors prove the disjointness of two strict stable classes in Datalog$^\neg$ programs only. In [27], the authors prove the disjointness of two strict supported classes in Datalog$^\neg$ programs only.

4.2 Relationships with the Stable and Supported Partial Model Semantics

We first establish a model-theoretic characterization for supported trap spaces in an NLP.

Theorem 4.3 (main result). *Given an NLP P, I is a supported trap space of P iff $I(\mathsf{rhs}_P(a)) \leq_s I(a)$ for every $a \in \mathsf{HB}_P$.*

Theorem 4.3 also points out that a supported trap space may not be a three-valued model of $\mathsf{comp}(P)$, as it considers the order $\leq_s$, whereas the completion considers the equivalence $\leftrightarrow$. Indeed, given the NLP P of Example 2.2, let I_4 be the three-valued interpretation where $I_4(a_0) = \star$ and $I_4(a) = 0$ for every $a \in \mathsf{HB}_P \setminus \{a_0\}$. It is easy to verify that I_4 is a supported trap space but not a three-valued model of $\mathsf{comp}(P)$.

To reason about stable trap spaces, we apply the least fixpoint transformation, which always produces a negative NLP. We then show that for negative NLPs, the stable transition graph coincides with the supported transition graph, entailing the coincidence between stable and supported trap spaces.

Theorem 4.4. *The stable and supported transition graphs of a negative NLP P are the same.*

Corollary 4.2. *Let P be a negative NLP. Then the set of stable trap spaces of P coincides with the set of supported trap spaces of P.*

Theorem 4.5 ([42])**.** *Given a negative NLP P, then the set of stable partial models of P coincides with the set of supported partial models of P.*

Theorem 4.6 (Theorem 3.1 of [3])**.** *Consider an NLP P. Let* $\mathsf{lfp}(P)$ *be the least fixpoint of P. Then P and* $\mathsf{lfp}(P)$ *have the same set of stable partial models, the same sets of regular models, and the same set of stable models.*

Based on the property that $F_P(I) = F_{\mathsf{lfp}(P)}(I)$ for any two-valued interpretation I, proved in [27], we directly obtain the following:

Theorem 4.7. *Let P be an NLP and* $\mathsf{lfp}(P)$ *denote the least fixpoint of P. Then* $\mathsf{tg}_{st}(P) = \mathsf{tg}_{st}(\mathsf{lfp}(P))$*, i.e., P and* $\mathsf{lfp}(P)$ *have the same stable transition graph.*

Since stable trap spaces are defined based on the stable transition graph, the above result immediately implies the following:

Corollary 4.3. *Let P be an NLP and* $\mathsf{lfp}(P)$ *denote the least fixpoint of P. Then P and* $\mathsf{lfp}(P)$ *have the same set of stable trap spaces.*

By Theorem 4.3, it is easy to derive the following simple relationship between supported partial models and supported trap spaces of an NLP:

Proposition 4.2. *Given an NLP P, if I is a supported partial model of P, then it is also a supported trap space of P.*

By applying the least fixpoint transformation, we obtain a similar relationship:

Proposition 4.3. *Given an NLP P, if I is a stable partial model of P, then it is also a stable trap space of P.*

Next, we establish a deeper relationship between supported trap spaces and supported partial models. We start with defining a new concept as follows:

Definition 4.1. *Consider an NLP P. We define a mapping* $f_P\colon \mathsf{SuTS}(P) \to \mathsf{SuTS}(P)$ *as follows: if m is a supported trap space, then* $f_P(m)(v) = m(\mathsf{rhs}_P(v)), \forall v \in \mathsf{HB}_P$*. The mapping* f_P *is well-defined because* $f_P(m)$ *is always a supported trap space if m is a supported trap space (see Lemma 4.3).*

We then show that the mapping f_P preserves the supported trap space property:

Lemma 4.3. *Given an NLP P and a supported trap space I of P,* $f_P(I)$ *is a supported trap space of P and* $I \leq_i f_P(I)$*.*

By combining Lemma 4.3 and the use of Hartogs' lemma (see Lemma 2.3), we obtain the following main result:

Theorem 4.8 (main result). *Consider an NLP P. For every supported trap space* $I \in \mathsf{SuTS}(P)$*, there is a unique supported partial model* $I' \in \mathsf{SuPM}(P)$ *such that* $I' \leq_s I$*.*

The beauty of Theorem 4.8 is that the function f_P delivers a unique supported partial model starting from any supported trap space. This insight is key to prove the following results.

Corollary 4.4. *Consider an NLP P. A three-valued interpretation I is a* $\leq_s$*-minimal supported partial model of P iff I is a* $\leq_s$*-minimal supported trap space of P.*

Corollary 4.4 is insightful as it indicates that, although the set of supported partial models may be different from the set of supported trap spaces, their $\leq_s$-minimal elements are the same. Hereafter, we show an interesting implication of the above relationships between supported trap spaces and supported partial models. We start by proving that a supported class that is included in a supported trap space is also included in the application of f_P to this supported trap space:

Lemma 4.4. *Consider an NLP P. Let I be a supported trap space of P. If S is a supported class of P such that $S \subseteq \mathscr{S}(I)$, then $S \subseteq \mathscr{S}(f_P(I))$.*

We now prove the case of supported classes:

Proposition 4.4. *Given an NLP P, if S is a supported class, then $\langle S \rangle_P^{sp}$ is a supported partial model.*

This is similar for stable classes:

Proposition 4.5. *Given an NLP P, if S is a stable class of P, then $\langle S \rangle_P^{st}$ is a stable partial model.*

Proposition 4.4 (resp. Proposition 4.5) gives us deeper understanding about relationships between the supported (resp. stable) classes and supported (resp. stable) partial models of an NLP.

4.3 Relationships with the Regular Model Semantics

Recall that a stable trap space may not be a stable partial model. Indeed, given the NLP P of Example 2.2, let I_4 be the three-valued interpretation where $I_4(a_0) = \star$ and $I_4(a) = 0$ for every $a \in \mathsf{HB}_P \setminus \{a_0\}$. It is easy to verify that I_4 is a stable trap space of P, but it is not a stable partial model of P. However, since every two-valued interpretation of P has an arc to I_1 in the stable transition graph of P, for every stable trap space I of P, $I(a) \neq 1$ for every $a \in \mathsf{HB}_P$. This implies that I_1 is the $\leq_s$-minimal stable trap space of P. Since I_1 is also the regular model of P, we wonder if the set of regular models of P coincides with the set of $\leq_s$-minimal stable trap spaces of P, which if true is really interesting. We answer this question for NLPs as follows:

Theorem 4.9 (main result). *Given an NLP P, a three-valued interpretation I is a regular model of P iff I is a $\leq_s$-minimal stable trap space of P.*

By Corollary 4.1, Proposition 3.3, and Theorem 4.9, each regular model of an NLP contains at least one strict stable class of this NLP, which is a new insight to the best of our knowledge.

4.4 Relationships with the L-stable Model Semantics

We first define a new order on $\mathbb{B}_\star$, which considers only the undefined value $\star$:

Definition 4.2. *The order $\leq_u$ on $\mathbb{B}_\star$ is given by $0 \leq_u 1$, $1 \leq_u 0$, $0 <_u \star$, $1 <_u \star$, and there is no other relation. Given two three-valued interpretations I_1 and I_2 of an NLP P, $I_1 \leq_u I_2$ iff $I_1(v) \leq_u I_2(v)$ for all $v \in \mathsf{HB}_P$. It is easy to see that $I_1 \leq_s I_2$ implies $I_1 \leq_u I_2$.*

Proposition 4.6. *Given two three-valued interpretations I_1 and I_2 of an NLP P, $I_1 \leq_u I_2$ iff $I_1^u \subseteq I_2^u$.*

Proposition 4.6 entails an equivalent definition for an L-stable model:

Corollary 4.5. *Given an NLP P, I is an L-stable model iff it is a $\leq_u$-minimal stable partial model.*

It is easy to derive that an L-stable model is also a regular model by Corollary 4.5 and the fact that $I_1 \leq_s I_2$ implies $I_1 \leq_u I_2$. However, the reverse might be not true in general. The relationship between $\leq_s$ and $\leq_u$ leads us to the following, which is similar to Theorem 4.8:

Proposition 4.7. *Consider an NLP P. For every supported trap space $I \in \mathsf{SuTS}(P)$, there is a supported partial model $J \in \mathsf{SuPM}(P)$ such that $J \leq_u I$.*

This property immediately implies that the $\leq_u$-minimal elements of the set of supported partial models and the set of supported trap spaces are the same.

Corollary 4.6. *Consider an NLP P. A three-valued interpretation I is a $\leq_u$-minimal supported partial model of P iff I is a $\leq_u$-minimal supported trap space of P.*

By using the least fixpoint transformation, we obtain the following equivalence:

Theorem 4.10. *Given an NLP P, I is an L-stable model iff I is a $\leq_u$-minimal stable trap space.*

Combining the above relationship, the least fixpoint transformation, and Theorem 4.9, we obtain the following:

Proposition 4.8. *Given an NLP P, I is an L-stable model of P iff I is a $\leq_u$-minimal regular model.*

The work by [35] shows that an L-stable model is a regular model. Proposition 4.8 provides a deeper relationship between the L-stable models and the regular models. Interestingly, this helps us to reveal the following insight:

Corollary 4.7. *If an NLP P has no L-stable model, then it has infinitely many regular models.*

5 Existence Results

Whether a given semantics is consistent (i.e., it contains at least one respective object) is a main property of interest [36, 38]. For Datalog$^\neg$ programs, proving or refuting this property is mostly straightforward [6]. However, for arbitrary NLPs with an infinite Herbrand base, conducting such proofs or refutations is more intricate. It often needs a precise use of set-theoretic axioms or equivalent statements. We here show that previous arguments for the existence of supported classes, strict supported (stable) classes, and regular models are either missing or imprecise; and that the trap space semantics can offer a unified and precise framework to prove the model or class existence. Furthermore, we reveal problems in the proof of [35] for the existence of L-stable models and correct it for uni-rule programs only (see more details in Section C).

We first recall a well-known result:

Theorem 5.1 (Theorem 1 of [4])**.** *Any NLP P has a stable class.*

The proof of Theorem 5.1 relies on the anti-monotonicity of F_P and the monotonicity of F_P^2 to construct two two-valued interpretations that form a stable class. However, this approach may not be directly applicable to supported classes, as T_P may be neither monotonic nor anti-monotonic. In [27, Proposition 5.6], the authors assert the guaranteed existence of stable and supported classes but do not provide formal proofs. While they state that "As explained in previous sections, existence of stable/supported classes is always guaranteed", our analysis indicates that this claim is not clearly substantiated by the preceding discussions in their work. Relying on the existence of $\subseteq$-minimal stable (resp. supported) trap sets, which can be easily derived from Lemma 4.1 (resp. Lemma 4.2), we provide an alternative proof for the existence of stable classes (resp. a new proof for the existence of supported classes) in an NLP.

Theorem 5.2. *An NLP P always has a $\subseteq$-minimal stable (or supported) trap set.*

Theorem 5.3. *Any NLP P has a supported class.*

While the existence of stable classes is precisely proved in [4], there has been no formal proof for the existence of strict stable (supported) classes so far. The work by [5] considers (finite) stable classes and suggests that verifying the existence of strict stable classes in such cases is straightforward, but even under these constraints, the derivation is not always evident indeed. Thanks to the relationships with $\subseteq$-minimal stable (resp. supported) trap sets established in the previous section, we can easily derive the existence of strict stable (resp. supported) classes:

Corollary 5.1 (**main result**). *An NLP P always has a strict stable (or supported) class.*

The equivalence between the regular models and the $\leq_s$-minimal stable trap spaces (see Theorem 4.9) and the existence of $\leq_s$-minimal stable trap spaces (see Theorem 3.1) immediately entail the existence of regular models in an NLP:

Corollary 5.2. *An NLP P always has a regular model.*

[42, Proposition 5.1] proves the existence of regular models in an NLP by showing that the NLP has at least one justifiable model (i.e., stable partial model) and that $<_{undef}$ (i.e.,$<_s$) is a quasi-order (i.e., a transitive and reflexive relation) on the set of justifiable models. However, this argument is only valid for finite sets, whereas the set of stable partial models of an NLP can be infinite. An infinite set can have no minimal element in general (see more examples in [38]). For infinite sets, it should follow the axiom of choice or equivalent statements [6]. With our new semantics, we can have an elegant proof for the existence of regular models that is precise w.r.t. the use of set-theoretic axioms.

Note that for NLPs, regular models are equivalent to M-stable models [19]. Since the existence of M-stable models is proved in [35, Corollary 5.4], the existence of regular models immediately follows. Hence, our proof of Corollary 5.2 offers an alternative (perhaps simpler) for the proof of [35, Corollary 5.4].

6 Conclusion

In this paper, we introduce the *trap space semantics* as a novel and general framework for interpreting NLPs. Extending prior work on Datalog$^\neg$ programs and their correspondence with Boolean networks, we generalize the concept of trap spaces to apply to arbitrary NLPs, including those with infinite Herbrand bases. This generalization provides a unified, semantics-based foundation that captures both model-theoretic and dynamical perspectives on NLPs.

We formally define the trap space semantics and established its foundational properties. Notably, we demonstrate that it encompasses and connects existing model-theoretic and dynamical semantics. Through this unification, the trap space semantics clarifies the relationships among these diverse approaches and provides a systematic means to compare and interpret them. See Figure 2 for a summary of the main relationships shown in Section 4.

Another significant contribution is the use of the trap space semantics as a unified framework for establishing existence results. We demonstrate how this framework enables rigorous proofs of the existence of supported classes, strict supported classes, strict stable classes, and regular models—results which were previously either absent or only informally stated in the literature.

More broadly, the trap space semantics advances the field by offering a conceptual bridge between the model-theoretic and dynamical views of NLPs. It allows us to reason simultaneously about fixed points of semantics-based operators and the dynamical behavior of NLPs over time. This duality opens up new avenues for understanding the behavior of NLPs comprehensively.

Looking forward, future research should focus on extending the trap space semantics to richer classes of programs, such as disjunctive logic programs, constraint logic programs, and extended logic programs.

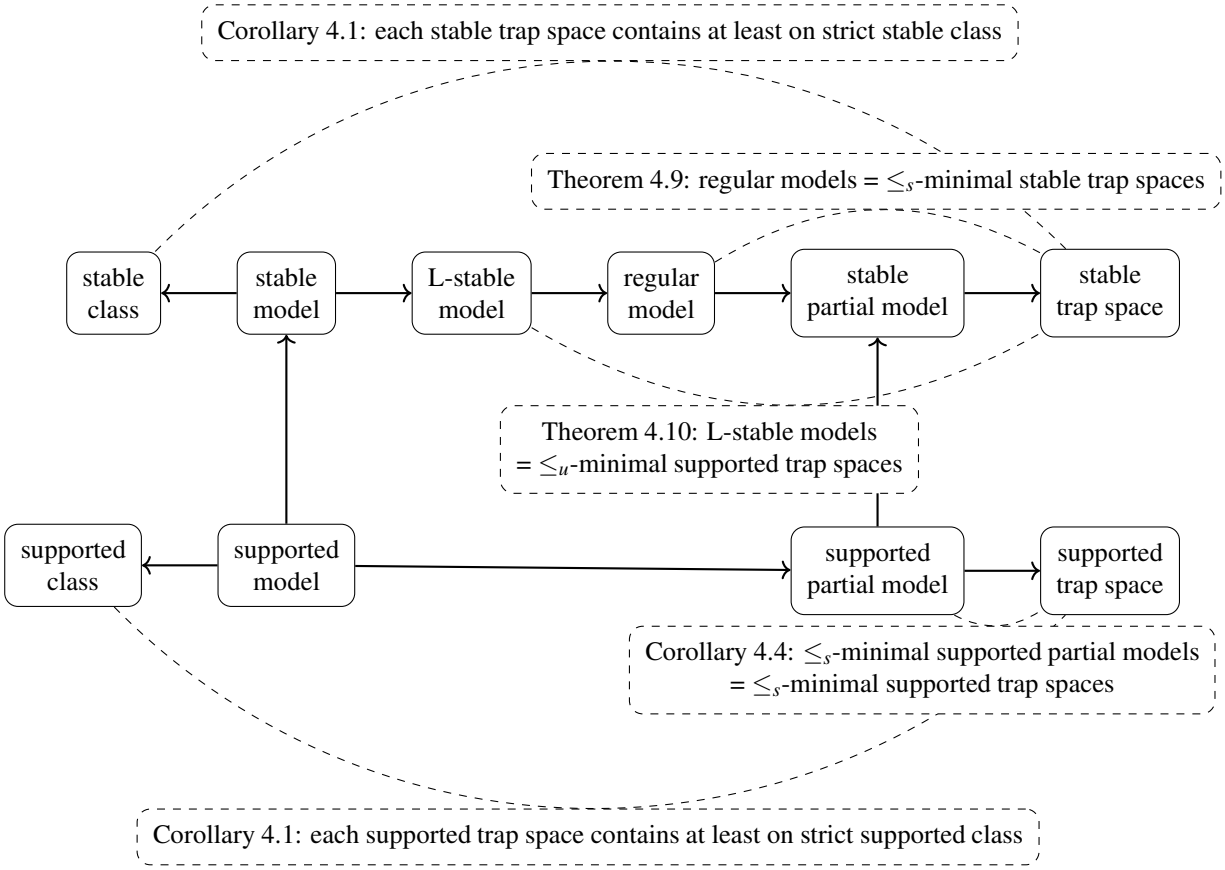


Figure 2: Summary of the main relationships shown in Section 4. A full arc indicates a set-inclusion relation, whereas a dashed line along with a box indicate the relationship between the two end vertices.

Furthermore, there are many other semantic proposals, for example, those that consider abstract properties such as cumulativity [15, 16]. Relating the trap space semantics to such semantics is interesting and can help clarify the semantic landscape. Finally, it is worth exploring the semantics' applications in areas that benefit from the interplay between logic and dynamics, including knowledge representation, verification, and systems biology.

References

[1] Krzysztof R Apt & Marc Bezem (1991): *Acyclic programs*. *New Gener. Comput.* 9, pp. 335–363, doi:10.1007/BF03037168.

[2] Krzysztof R Apt, Howard A Blair & Adrian Walker (1988): *Towards a theory of declarative knowledge*. In: *Foundations of Deductive Databases and Logic Programming*, Elsevier, San Francisco, pp. 89–148.

[3] Chandrabose Aravindan & Phan Minh Dung (1995): *On the Correctness of Unfold/Fold Transformation of Normal and Extended Logic Programs*. *J. Log. Program.* 24(3), pp. 201–217, doi:10.1016/0743-1066(94)00104-E.

[4] Chitta Baral & V. S. Subrahmanian (1992): *Stable and Extension Class Theory for Logic Programs and Default Logics*. *J. Autom. Reason.* 8(3), pp. 345–366, doi:10.1007/BF02341854.

[5] Chitta Baral & V. S. Subrahmanian (1993): *Dualities Between Alternative Semantics for Logic Programming and Nonmonotonic Reasoning*. *J. Autom. Reason.* 10(3), pp. 399–420, doi:10.1007/BF00881799.

[6] Ringo Baumann & Christof Spanring (2015): *Infinite argumentation frameworks: On the existence and uniqueness of extensions*. In: *Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation: Essays Dedicated to Gerhard Brewka on the Occasion of his 60th Birthday*, Springer, Cham, pp. 281–295, doi:10.1007/978-3-319-14726-0_19.

[7] Belaid Benhamou & Pierre Siegel (2012): *A New Semantics for Logic Programs Capturing and Extending the Stable Model Semantics*. In: *Proc. of ICTAI*, IEEE Computer Society, Athens, Greece, pp. 572–579, doi:10.1109/ICTAI.2012.167.

[8] Howard A. Blair, Fred Dushin & Paul R. Humenn (1997): *Simulations between Programs as Cellular Automata*. In: *Proc. of LPNMR*, Springer, Berlin, Heidelberg, pp. 115–131, doi:10.1007/3-540-63255-7_9.

[9] Martin Caminada, Samy Sá, João F. L. Alcântara & Wolfgang Dvorák (2015): *On the equivalence between logic programming semantics and argumentation semantics*. *Int. J. Approx. Reason.* 58, pp. 87–111, doi:10.1016/J.IJAR.2014.12.004.

[10] Martin Caminada & Claudia Schulz (2017): *On the Equivalence between Assumption-Based Argumentation and Logic Programming*. *J. Artif. Intell. Res.* 60, pp. 779–825, doi:10.1613/JAIR.5581.

[11] Martin W. A. Caminada & Dov M. Gabbay (2009): *A Logical Account of Formal Argumentation*. *Stud Logica* 93(2-3), pp. 109–145, doi:10.1007/S11225-009-9218-X.

[12] Stefano Ceri, Georg Gottlob & Letizia Tanca (1990): *Logic Programming and Databases: An Overview*. Springer, Berlin, doi:10.1007/978-3-642-83952-8.

[13] Keith L. Clark (1977): *Negation as Failure*. In: *Logic and Data Bases, Symposium on Logic and Data Bases*, Plemum Press, New York, pp. 293–322, doi:10.1007/978-1-4684-3384-5_11.

[14] Emmanuelle-Anna Dietz, Steffen Hölldobler & Christoph Wernhard (2014): *Modeling the suppression task under weak completion and well-founded semantics*. *J. Appl. Non Class. Logics* 24(1-2), pp. 61–85, doi:10.1080/11663081.2014.911520.

[15] Jürgen Dix (1995): *A Classification Theory of Semantics of Normal Logic Programs: I. Strong Properties*. *Fundam. Informaticae* 22(3), pp. 227–255, doi:10.3233/FI-1995-2233.

[16] Jürgen Dix (1995): *A Classification Theory of Semantics of Normal Logic Programs: II. Weak Properties*. *Fundam. Informaticae* 22(3), pp. 257–288, doi:10.3233/FI-1995-2234.

[17] Phan Minh Dung (1995): *On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games*. *Artif. Intell.* 77(2), pp. 321–358, doi:10.1016/0004-3702(94)00041-X.

[18] Phan Minh Dung & Kanchana Kanchanasut (1989): *A Fixpoint Approach to Declarative Semantics of Logic Programs*. In: *Proc. of NACLP*, MIT Press, Cleveland, Ohio, pp. 604–625.

[19] Thomas Eiter, Nicola Leone & Domenico Saccà (1997): *On the Partial Semantics for Disjunctive Deductive Databases*. *Ann. Math. Artif. Intell.* 19(1-2), pp. 59–96, doi:10.1023/A:1018947420290.

[20] François Fages (1994): *Consistency of Clark's completion and existence of stable models*. *Methods Log. Comput. Sci.* 1(1), pp. 51–60.

[21] François Fages (1991): *A New Fixpoint Semantics for General Logic Programs Compared with the Well-Founded and the Stable Model Semantics*. *New Gener. Comput.* 9(3/4), pp. 425–444, doi:10.1007/BF03037172.

[22] Allen Van Gelder, Kenneth A. Ross & John S. Schlipf (1988): *Unfounded Sets and Well-Founded Semantics for General Logic Programs*. In: *Proc. of ACM SIGACT-SIGMOD-SIGART*, ACM, Austin, Texas, pp. 221–230, doi:10.1145/308386.308444.

[23] Allen Van Gelder, Kenneth A. Ross & John S. Schlipf (1991): *The Well-Founded Semantics for General Logic Programs*. *J. ACM* 38(3), pp. 620–650, doi:10.1145/116825.116838.

[24] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In: *Proc. of ICLP/SLP*, MIT Press, Seattle, pp. 1070–1080.

[25] Paul Richard Halmos (1960): *Naive set theory*. van Nostrand, New York, doi:10.1007/978-1-4757-1645-0.

[26] F. Hartogs (1915): *Über das Problem der Wohlordnung*. *Mathematische Annalen* 76(4), p. 438–443, doi:10.1007/bf01458215.

[27] Katsumi Inoue & Chiaki Sakama (2012): *Oscillating Behavior of Logic Programs*. In: *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, Springer, Berlin, pp. 345–362, doi:10.1007/978-3-642-30743-0_23.

[28] Hannes Klarner, Alexander Bockmayr & Heike Siebert (2015): *Computing maximal and minimal trap spaces of Boolean networks*. *Nat. Comput.* 14(4), pp. 535–544, doi:10.1007/S11047-015-9520-7.

[29] Dénes König (1990): *Theory of finite and infinite graphs*. In: *Theory of Finite and Infinite Graphs*, Springer, Birkhäuser Boston, MA, pp. 45–421, doi:10.1007/978-1-4684-8971-2.

[30] John Wylie Lloyd (1984): *Foundations of Logic Programming*. Springer, Berlin Heidelberg, doi:10.1007/978-3-642-96826-6.

[31] V. Wiktor Marek & V. S. Subrahmanian (1992): *The Relationship Between Stable, Supported, Default and Autoepistemic Semantics for General Logic Programs*. *Theor. Comput. Sci.* 103(2), pp. 365–386, doi:10.1016/0304-3975(92)90019-C.

[32] David Pearce (2006): *Equilibrium logic*. *Ann. Math. Artif. Intell.* 47(1), pp. 3–41, doi:10.1007/s10472-006-9028-z.

[33] Teodor C. Przymusinski (1990): *The Well-Founded Semantics Coincides with the Three-Valued Stable Semantics*. *Fundam. Inform.* 13(4), pp. 445–463, doi:10.3233/FI-1990-13404.

[34] Teodor C. Przymusinski (1994): *Well-founded and Stationary Models of Logic Programs*. *Ann. Math. Artif. Intell.* 12(3-4), pp. 141–187, doi:10.1007/BF01530784.

[35] Domenico Saccà & Carlo Zaniolo (1997): *Deterministic and Non-Deterministic Stable Models*. *J. Log. Comput.* 7(5), pp. 555–579, doi:10.1093/LOGCOM/7.5.555.

[36] Taisuke Sato (1990): *Completed Logic Programs and their Consistency*. *J. Log. Program.* 9(1), pp. 33–44, doi:10.1016/0743-1066(90)90032-Z.

[37] J. Seitzer & John S. Schlipf (1997): *Affordable Classes of Normal Logic Programs*. In: *Proc. of LPNMR*, Springer, Berlin, Heidelberg, pp. 92–111, doi:10.1007/3-540-63255-7_7.

[38] Christof Spanring (2017): *Set- and graph-theoretic investigations in abstract argumentation*. Ph.D. thesis, University of Liverpool, UK. Available at https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.755437.

[39] Van-Giang Trinh, Belaid Benhamou & Sylvain Soliman (2023): *Trap spaces of Boolean networks are conflict-free siphons of their Petri net encoding*. *Theor. Comput. Sci.* 971, p. 114073, doi:10.1016/j.tcs.2023.114073.

[40] Van-Giang Trinh, Belaid Benhamou, Sylvain Soliman & François Fages (2025): *On the Boolean Network Theory of Datalog¬*. arXiv:2504.15417. https://arxiv.org/abs/2504.15417.

[41] Yining Wu, Martin Caminada & Dov M. Gabbay (2009): *Complete Extensions in Argumentation Coincide with 3-Valued Stable Models in Logic Programming*. *Stud Logica* 93(2-3), pp. 383–403, doi:10.1007/S11225-009-9210-5.

[42] Jia-Huai You & Li-Yan Yuan (1994): *A Three-Valued Semantics for Deductive Databases and Logic Programs*. *J. Comput. Syst. Sci.* 49(2), pp. 334–361, doi:10.1016/S0022-0000(05)80053-4.

[43] Jia-Huai You & Li-Yan Yuan (1995): *On the Equivalence of Semantics for Normal Logic Programs*. *J. Log. Program.* 22(3), pp. 211–222, doi:10.1016/0743-1066(94)00023-Y.

[44] Max Zorn (1935): *A remark on method in transfinite algebra*. *Bull. Am. Math. Soc.* 41(10), pp. 667–670.

A Detailed Proofs

This appendix shows the detailed proofs of the results shown in the main text of the present paper.

Proposition 3.1. *An NLP P always has a stable or supported trap space.*

Proof. Let I^* be a three-valued interpretation that corresponds to all the two-valued interpretations, i.e., $\forall a \in \mathsf{HB}_P, I^*(a) = \star$. By setting $S = \mathscr{S}(I^*)$, the condition $\{F_P(I) \mid I \in S\} \subseteq S$ (resp. $\{T_P(I) \mid I \in S\} \subseteq S$) always holds. Hence, $\mathscr{S}(I^*)$ is a stable (resp. supported) trap set of P. By definition, I^* is a stable (resp. supported) trap space of P. □

Lemma 3.1. *Consider an NLP P and a three-valued interpretation J of P. If J is a stable trap space of P, then there exists a $\leq_s$-minimal stable trap space J' of P such that $J' \leq_s J$.*

Proof. Clearly, $(\mathsf{StTS}(P), \leq_i)$ is a partially ordered set. Consider a $\leq_i$-chain $(E_k)_{k\in I}$. Note that I is the set of all elements of the $\leq_i$-chain, and it can be written as an indexed family, which can be done via using the set itself as index set. Consider now the intersection of all elements in I, said $\sqcap_I$ whose existence is guaranteed by Definition 3.5. Following from the definition, we have $\mathscr{S}(\sqcap_I) = \bigcap_{k\in I} \mathscr{S}(E_k)$. It follows that $\sqcap_I \leq_s E_k$ for all $E_k, k \in I$. Equivalently, $E_k \leq_i \sqcap_I$ for all $E_k, k \in I$, implying that $\sqcap_I$ is an upper bound w.r.t. $\leq_i$ of $(E_k)_{k\in I}$. It remains to show that $\sqcap_I$ is a stable trap space of P.

Consider a two-valued interpretation A of P. Let B be the successor of A in $\mathsf{tg}_{st}(P)$, i.e., $B = F_P(A)$. Assume that $A \in \mathscr{S}(\sqcap_I)$. Then for all $k \in I$, we have that $A \in \mathscr{S}(E_k)$. Thus $B \in \mathscr{S}(E_k)$, since E_k is a stable trap space. Hence, $B \in \mathscr{S}(\sqcap_I)$. This implies that if $A \in \mathscr{S}(\sqcap_I)$ then $F_P(A) \in \mathscr{S}(\sqcap_I)$, thus $\sqcap_I$ is a stable trap set of P. Since $\sqcap_I$ is a three-valued interpretation by definition, it is a stable trap space of P.

By definition, a $\leq_s$-minimal stable trap space is equivalent to a $\leq_i$-maximal stable trap space. By Lemma 2.2, if J is a stable trap space of P, then there exists a $\leq_s$-minimal stable trap space J' of P such that $J \leq_i J'$ (equivalently, $J' \leq_s J$). □

Lemma 3.2. *Consider an NLP P and a three-valued interpretation J of P. If J is a supported trap space of P, then there exists a $\leq_s$-minimal supported trap space J' of P such that $J' \leq_s J$.*

Proof. The proof of Lemma 3.2 can be applied here by substituting $F_P(A)$ in Lemma 3.1 with $T_P(A)$. □

Theorem 3.1 (main result). *Any NLP P has a $\leq_s$-minimal stable (or supported) trap space.*

Proof. For the case of $\leq_s$-minimal stable trap spaces, this immediately follows from Lemma 3.1 and the fact that P has at least one stable trap space by Proposition 3.1.

For the case of $\leq_s$-minimal supported trap spaces, this immediately follows from Lemma 3.2 and the fact that P has at least one supported trap space by Proposition 3.1. □

Proposition 3.2. *Consider an NLP P. Let S be a non-empty set of two-valued interpretations of P. Then there is a unique $\leq_s$-minimal stable (resp. supported) trap space, denoted by $\langle S\rangle_P^{st}$ (resp. $\langle S\rangle_P^{sp}$), covering S, i.e., $S \subseteq \mathscr{S}(\langle S\rangle_P^{st})$ (resp. $S \subseteq \mathscr{S}(\langle S\rangle_P^{sp})$).*

Proof. Hereafter, we prove the case of stable trap spaces. The proof for the case of supported trap spaces is analogous. Let T be the set of all stable trap spaces I such that $S \subseteq \mathscr{S}(I)$. The set T is non-empty as it contains ε that is a stable trap space in which all atoms are assigned to $\star$ (see Proposition 3.1). Since S is non-empty, the elements in T are mutually consistent, thus we can take the intersection of all these elements said $\sqcap_T$.

Next, we show that $\sqcap_T$ is also a stable trap space of P. Consider a two-valued interpretation A of P. Let B be the successor of A in $\mathtt{tg}_{st}(P)$, i.e., $B = F_P(A)$. Assume that $A \in \mathscr{S}(\sqcap_T)$. Then for all $E_k \in T$, we have that $A \in \mathscr{S}(E_k)$. Thus $B \in \mathscr{S}(E_k)$, since E_k is a stable trap space. Hence, $B \in \mathscr{S}(\sqcap_T)$. This implies that if $A \in \mathscr{S}(\sqcap_T)$ then $F_P(A) \in \mathscr{S}(\sqcap_T)$, thus $\sqcap_T$ is a stable trap set of P. Since $\sqcap_T$ is a three-valued interpretation by definition, it is a stable trap space of P.

By definition, $\sqcap_T$ is unique and $\leq_s$-minimal. Since $S \subseteq \mathscr{S}(E_k)$ for all $E_k \in T$, $S \subseteq \mathscr{S}(\sqcap_T)$. By setting $\langle S \rangle_P^{st} = \sqcap_T$, we can conclude the proof. □

Proposition 3.3. *Consider an NLP P. Let I_1 and I_2 be two distinct $\leq_s$-minimal stable (resp. supported) trap spaces of P. Then I_1 and I_2 are not consistent.*

Proof. Hereafter, we prove the case of stable trap spaces. The proof for the case of supported trap spaces is analogous. Assume that I_1 and I_2 are consistent. Then $I = \sqcap_{\{I_1, I_2\}}$ exists and $\mathscr{S}(I) = \mathscr{S}(I_1) \cap \mathscr{S}(I_2)$. By similarly applying the arguments of Proposition 3.2, we have I is also a stable trap space of P. Since I_1 and I_2 are distinct, we have $\mathscr{S}(I_1) \subset \mathscr{S}(I)$ or $\mathscr{S}(I_2) \subset \mathscr{S}(I)$ must hold. This leads to a contradiction because I_1 and I_2 are $\leq_s$-minimal stable trap spaces of P. Hence, I_1 and I_2 are not consistent. □

Lemma 4.1. *Consider an NLP P and a non-empty set S of two-valued interpretations of P. If S is a stable trap set of P, then there exists a $\subseteq$-minimal stable trap set S' of P such that $S' \subseteq S$.*

Proof. Let $TS(P)$ denote the set of all stable trap sets of P. Clearly, $(TS(P), \supseteq)$ is a partially ordered set. Given a $\supseteq$-chain $(E_i)_{i \in I}$. Note that I is the set of all elements of the $\supseteq$-chain, and it can be written as an indexed family, which can be done via using the set itself as index set. Consider now $E = \bigcap_{i \in I} E_i$. Obviously, E is an upper bound of $(E_i)_{i \in I}$, i.e., $E_i \supseteq E$, for all $i \in I$. It remains to show that E is a stable trap set of P.

Consider a two-valued interpretation A. $A \in E$ iff $A \in E_i, \forall i \in I$. This implies that $F_P(A) \in E_i, \forall i \in I$ since E_i is a stable trap set of P. Then $F_P(A) \in E$. This implies that $\{F_P(B) \mid B \in E\} \subseteq E$. By definition, E is a stable trap set of P.

By definition, $\subseteq$-minimal stable trap set is equivalent to a $\supseteq$-maximal stable trap set. By Lemma 2.2, if S is a stable trap set of P, then there exists a $\subseteq$-minimal stable trap set S' of P such that $S \supseteq S'$ (equivalently, $S' \subseteq S$). □

Lemma 4.2. *Given an NLP P, if a non-empty set S of two-valued interpretations of P is a supported trap set of P, then there exists a $\subseteq$-minimal supported trap set S' of P such that $S' \subseteq S$.*

Proof. The proof of Lemma 4.1 can be applied here by substituting $F_P(I)$ in Lemma 4.1 with $T_P(I)$. □

Theorem 4.1 (main result). *Consider an NLP P. A non-empty set S of two-valued interpretations is a $\subseteq$-minimal stable trap set of P iff S is a strict stable class of P.*

Proof. "⇒" Assume that S is a $\subseteq$-minimal stable trap set of P. Then $\{F_P(I) \mid I \in S\} \subseteq S$ by definition. Suppose that $\{F_P(I) \mid I \in S\} \subsetneq S$. Then there exists $J \in S$ such that for all $I \in S$, $F_P(I) \neq J$. We have $\{F_P(I) \mid I \in S \setminus \{J\}\} \subseteq \{F_P(I) \mid I \in S\} \subseteq S \setminus \{J\}$. Thus $S \setminus \{J\}$ is a stable trap set of P, which contradicts the $\subseteq$-minimality of S. Hence $\{F_P(I) \mid I \in S\} = S$, i.e., S is a stable class of P. Since a stable class is a stable trap set, S must be a strict stable class.

"⇐" Assume that S is a strict stable class of P. Then $\{F_P(I) \mid I \in S\} = S$ by definition. This implies that S is a stable trap set of P. By Lemma 4.1, there is a $\subseteq$-minimal stable trap set S' such that $S' \subseteq S$. We have $\{F_P(I) \mid I \in S'\} \subseteq S'$ by definition. Suppose that $\{F_P(I) \mid I \in S'\} \subsetneq S'$. Then there is $J \in S'$ such

that $F_P(I) \neq J$ for all $I \in S'$. We have $\{F_P(I) \mid I \in S' \setminus \{J\}\} \subseteq \{F_P(I) \mid I \in S'\} \subseteq S' \setminus \{J\}$. By definition, $S' \setminus \{J\}$ is a stable trap set of P, which contradicts the $\subseteq$-minimality of S'. Thus, $\{F_P(I) \mid I \in S'\} = S'$, i.e., S' is a stable class of P. S is a strict stable class of P, thus $S = S'$ because of the $\subseteq$-minimality of a strict stable class. Hence, S is a $\subseteq$-minimal stable trap set of P. □

Theorem 4.2. *Consider an NLP P. A non-empty set S of two-valued interpretations is a $\subseteq$-minimal supported trap set of P iff S is a strict supported class of P.*

Proof. The proof of Theorem 4.1 can be applied here by substituting $F_P(I)$ and Lemma 4.1 in Theorem 4.1 with $T_P(I)$ and Lemma 4.2, respectively. □

Corollary 4.1. *Consider an NLP P. For every stable (resp. supported) trap space I of P, $\mathscr{S}(I)$ contains at least one strict stable (resp. supported) class of P.*

Proof. Assume that I is a stable trap space of P. By definition, $\mathscr{S}(I)$ is a stable trap set of P. There exists a $\subseteq$-minimal stable trap set $S' \subseteq \mathscr{S}(I)$. By Theorem 4.1, S' is also a strict stable class of P.

Assume that I is a supported trap space of P. By definition, $\mathscr{S}(I)$ is a supported trap set of P. There exists a $\subseteq$-minimal supported trap set $S' \subseteq \mathscr{S}(I)$. By Theorem 4.2, S' is also a strict supported class of P. □

Proposition 4.1. *Two distinct strict stable (resp. supported) classes of an NLP P are disjoint.*

Proof. We prove the case of strict stable classes first.

Let S_1 and S_2 be two distinct strict stable classes of P. By Theorem 4.1, S_1 and S_2 are two distinct $\subseteq$-minimal stable trap sets of P. Assume that S_1 and S_2 are not disjoint. It follows that $S_1 \cap S_2 \neq \emptyset$. Consider $I \in S_1 \cap S_2$. We have $I \in S_1$ and $I \in S_2$, then $F_P(I) \in S_1$ and $F_P(I) \in S_2$ because S_1 and S_2 are stable trap sets, thus $F_P(I) \in S_1 \cap S_2$. It follows that $S_1 \cap S_2$ is a stable trap set of P. Then $S_1 \cap S_2 \subsetneq S_1$ or $S_1 \cap S_2 \subsetneq S_2$ must hold because $S_1 \neq S_2$, which contradicts the $\subseteq$-minimality of S_1 and S_2. Hence, S_1 and S_2 are disjoint.

For the case of strict supported classes, we only need to replace F_P and Theorem 4.1 with T_P and Theorem 4.2, respectively. □

Theorem 4.3 (main result). *Given an NLP P, I is a supported trap space of P iff $I(\mathsf{rhs}_P(a)) \leq_s I(a)$ for every $a \in \mathsf{HB}_P$.*

Proof. "⇒" Assume that I is a supported trap space of P. Suppose that there is an atom $a \in \mathsf{HB}_P$ such that $I(a) <_s I(\mathsf{rhs}_P(a))$. By the definition of the order $\leq_s$, we have $I(\mathsf{rhs}_P(a)) = \star$ and $I(a) \neq \star$. Since $I(\mathsf{rhs}_P(a)) = \star$, there exists a two-valued interpretation $J \in \mathscr{S}(I)$ such that $J(\mathsf{rhs}_P(a)) = \neg I(a)$. By definition, $T_P(J)(a) = J(\mathsf{rhs}_P(a))$, thus $T_P(J)$ does not belong to $\mathscr{S}(I)$. This implies that $\mathscr{S}(I)$ is not a supported trap set by definition, which is a contradiction. Hence, $I(\mathsf{rhs}_P(a)) \leq_s I(a)$ for every $a \in \mathsf{HB}_P$.

"⇐" Assume that $I(\mathsf{rhs}_P(a)) \leq_s I(a)$ for every atom $a \in \mathsf{HB}_P$. Let A be a two-valued interpretation in $\mathscr{S}(I)$. Of course, $A \leq_s I$. By the characteristics of the three-valued logic, we have $A(\mathsf{rhs}_P(a)) \leq_s I(\mathsf{rhs}_P(a))$, then $A(\mathsf{rhs}_P(a)) \leq_s I(a)$ for every $a \in \mathsf{HB}_P$. This implies that $T_P(A) \leq_s I$, thus $T_P(A) \in \mathscr{S}(I)$. Hence, $\mathscr{S}(I)$ is a supported trap set of P. By definition, I is a supported trap space of P. □

Theorem 4.4. *The stable and supported transition graphs of a negative NLP P are the same.*

Proof. It has been shown in [27] that for any two-valued interpretation I, $F_P(I) = T_P(I)$, if P is a negative NLP. By the definitions of stable and supported transition graphs, we have that $\mathsf{tg}_{st}(P) = \mathsf{tg}_{sp}(P)$. □

Corollary 4.2. *Let P be a negative NLP. Then the set of stable trap spaces of P coincides with the set of supported trap spaces of P.*

Proof. This immediately follows from Theorem 4.4. □

Theorem 4.7. *Let P be an NLP and $\mathsf{lfp}(P)$ denote the least fixpoint of P. Then $\mathsf{tg}_{st}(P) = \mathsf{tg}_{st}(\mathsf{lfp}(P))$, i.e., P and $\mathsf{lfp}(P)$ have the same stable transition graph.*

Proof. It has been shown in [27] that for any two-valued interpretation I, we have that $F_P(I) = F_{\mathsf{lfp}(P)}(I)$. By the definition of stable transition graph, we have that $\mathsf{tg}_{st}(P) = \mathsf{tg}_{st}(\mathsf{lfp}(P))$. □

Corollary 4.3. *Let P be an NLP and $\mathsf{lfp}(P)$ denote the least fixpoint of P. Then P and $\mathsf{lfp}(P)$ have the same set of stable trap spaces.*

Proof. This immediately follows from Theorem 4.7. □

Proposition 4.2. *Given an NLP P, if I is a supported partial model of P, then it is also a supported trap space of P.*

Proof. Assume that I is a supported partial model of P. Then I is a three-valued model of $\mathsf{comp}(P)$. It follows that $I(a) = I(\mathsf{rhs}_P(a))$, $\forall a \in \mathsf{HB}_P$. Thus, I also satisfies the condition $I(\mathsf{rhs}_P(a)) \leq_s I(a)$, $\forall a \in \mathsf{HB}_P$. By Theorem 4.3, I is also a supported trap space of P. □

Proposition 4.3. *Given an NLP P, if I is a stable partial model of P, then it is also a stable trap space of P.*

Proof. Assume that I is a stable partial model of P.
Then I is also a stable partial model of $\mathsf{lfp}(P)$ by Theorem 4.6.
Since $\mathsf{lfp}(P)$ is a negative NLP, I is also a supported partial model of $\mathsf{lfp}(P)$ by Theorem 4.5.
By Proposition 4.2, I is a supported trap space of $\mathsf{lfp}(P)$.
By Corollary 4.2, I is also a stable trap space of $\mathsf{lfp}(P)$.
By Corollary 4.3, I is a stable trap space of P. □

Lemma 4.3. *Given an NLP P and a supported trap space I of P, $f_P(I)$ is a supported trap space of P and $I \leq_i f_P(I)$.*

Proof. By definition, $f_P(I)(v) = I(\mathsf{rhs}_P(v))$ for all $v \in \mathsf{HB}_P$. Since I is a supported trap space of P, $I(v) \leq_i I(\mathsf{rhs}_P(v))$ for all $v \in \mathsf{HB}_P$ by Theorem 4.3. Thus, $I \leq_i f_P(I)$. Then, by the truth table of the three-valued logic, $I(\mathsf{rhs}_P(v)) \leq_i f_P(I)(\mathsf{rhs}_P(v))$ for all $v \in \mathsf{HB}_P$. This implies that $f_P(I)(v) \leq_i f_P(I)(\mathsf{rhs}_P(v))$ for all $v \in \mathsf{HB}_P$. Hence, $f_P(I)$ is a supported trap space of P by Theorem 4.3. □

Theorem 4.8 (main result). *Consider an NLP P. For every supported trap space $I \in \mathsf{SuTS}(P)$, there is a unique supported partial model $I' \in \mathsf{SuPM}(P)$ such that $I' \leq_s I$.*

Proof. We define a function K recursively on the ordinals as follows:

$$K(0) = I$$
$$K(\alpha+1) = f_P(K(\alpha)).$$

If β is a limit ordinal, then by construction and Lemma 4.3 $\{K(\alpha)\colon \alpha < \beta\}$ is a chain w.r.t. $\leq_i$ in $\mathsf{SuTS}(P)$. Define $K(\beta) = \sup_{\leq_i}\{K(\alpha)\colon \alpha < \beta\}$, where $\sup_{\leq_i}$ denotes the least upper bound w.r.t. $\leq_i$.

We have K is an increasing function from the ordinals into $\mathsf{SuTS}(P)$. It cannot be strictly increasing, as if it was, then we would have an injection from the ordinals into $\mathsf{SuTS}(P)$, which violates Lemma 2.3. Hence, K must be eventually constant. It follows that for some ordinal α, $K(\alpha+1) = K(\alpha)$, then $f_P(K(\alpha)) = K(\alpha)$. Let α^* be the least ordinal such that $K(\alpha^*+1) = K(\alpha^*)$. We have $K(\beta) = K(\alpha^*)$ for all $\beta > \alpha^*$. Hence, $K(\alpha^*)$ is unique. Set $I' = K(\alpha^*)$. We have $f_P(I') = I'$, i.e., I' is a three-valued model of $\mathsf{comp}(P)$. Therefore, $I' \in \mathsf{SuPM}(P)$. By construction and Lemma 4.3, $I \leq_i I'$. This implies that $I' \leq_s I$. □

Corollary 4.4. *Consider an NLP P. A three-valued interpretation I is a $\leq_s$-minimal supported partial model of P iff I is a $\leq_s$-minimal supported trap space of P.*

Proof. Recall that $\mathsf{SuTS}(P)$ (resp. $\mathsf{SuPM}(P)$) denotes the set of all supported trap spaces (resp. partial models) of P. We have $\mathsf{SuPM}(P) \subseteq \mathsf{SuTS}(P)$ by Proposition 4.2.

"⇒" Assume that I is not a $\leq_s$-minimal supported trap space of P. Then there exists $I' \in \mathsf{SuTS}(P)$ such that $I' <_s I$. By Theorem 4.8, there exists $J \in \mathsf{SuPM}(P)$ such that $J \leq_s I'$. This implies that $J <_s I$, which contradicts the $\leq_s$-minimality of I. Hence, I is a $\leq_s$-minimal supported trap space of P.

"⇐" This direction is trivial because $\mathsf{SuPM}(P) \subseteq \mathsf{SuTS}(P)$. □

Lemma 4.4. *Consider an NLP P. Let I be a supported trap space of P. If S is a supported class of P such that $S \subseteq \mathscr{S}(I)$, then $S \subseteq \mathscr{S}(f_P(I))$.*

Proof. Consider $J \in S$. Since $S \subseteq \mathscr{S}(I)$, we have $J \in \mathscr{S}(I)$. Let $J' = T_P(J)$. For any atom $a \in \mathsf{HB}_P$, we have $J'(a) = J(\mathsf{rhs}_P(a)) \leq_s I(\mathsf{rhs}_P(a)) = f_P(I)(a)$. This implies that $J' \in \mathscr{S}(f_P(I))$. Hence, $\{T_P(J) \mid J \in S\} \subseteq \mathscr{S}(f_P(I))$. Since S is a supported class, $\{T_P(J) \mid J \in S\} = S$ by definition. This implies that $S \subseteq \mathscr{S}(f_P(I))$. □

Proposition 4.4. *Given an NLP P, if S is a supported class, then $\langle S\rangle_P^{sp}$ is a supported partial model.*

Proof. By Proposition 3.2, $\langle S\rangle_P^{sp}$ is a supported trap space of P. Assume that it is not a supported partial model of P. Then by definition, we have $f_P(\langle S\rangle_P^{sp}) \neq \langle S\rangle_P^{sp}$. By Lemma 4.3, $f_P(\langle S\rangle_P^{sp}) \leq_s \langle S\rangle_P^{sp}$. It follows that $f_P(\langle S\rangle_P^{sp}) <_s \langle S\rangle_P^{sp}$. By Lemma 4.4, $S \subseteq \mathscr{S}(f_P(\langle S\rangle_P^{sp}))$ because S is a supported class and $S \subseteq \mathscr{S}(\langle S\rangle_P^{sp})$ by definition. This implies that $\langle S\rangle_P^{sp}$ is not a $\leq_s$-minimal supported trap space covering S, which is a contradiction. Hence, $\langle S\rangle_P^{sp}$ is a supported partial model of P. □

Proposition 4.5. *Given an NLP P, if S is a stable class of P, then $\langle S\rangle_P^{st}$ is a stable partial model.*

Proof. By Theorem 4.7 and Theorem 4.4, $\mathsf{tg}_{st}(P) = \mathsf{tg}_{st}(\mathsf{lfp}(P)) = \mathsf{tg}_{sp}(\mathsf{lfp}(P))$. Hence, S is a supported class of $\mathsf{lfp}(P)$. In addition, $\langle S\rangle_P^{st} = \langle S\rangle_{\mathsf{lfp}(P)}^{sp}$. By Proposition 4.4, $\langle S\rangle_{\mathsf{lfp}(P)}^{sp}$ is a supported partial model of $\mathsf{lfp}(P)$. It is also a stable partial model of $\mathsf{lfp}(P)$ by Theorem 4.5, since $\mathsf{lfp}(P)$ is negative. By Theorem 4.6, $\langle S\rangle_P^{st}$ is a stable partial model of P. □

Theorem 4.9 (main result). *Given an NLP P, a three-valued interpretation I is a regular model of P iff I is a $\leq_s$-minimal stable trap space of P.*

Proof. The programs P and $\mathsf{lfp}(P)$ have the same set of regular models by Theorem 4.6. They also have the same set of stable trap spaces by Corollary 4.3. Hence, it suffices to show that I is a regular model of $\mathsf{lfp}(P)$ iff I is a $\leq_s$-minimal stable trap space of $\mathsf{lfp}(P)$.

Recall that $\mathsf{lfp}(P)$ is a negative NLP. Then I is a regular model of $\mathsf{lfp}(P)$
iff I is a $\leq_s$-minimal stable partial model of $\mathsf{lfp}(P)$ by definition

iff I is a $\leq_s$-minimal supported partial model of $\mathsf{lfp}(P)$ by Theorem 4.5
iff I is a $\leq_s$-minimal supported trap space of $\mathsf{lfp}(P)$ by Corollary 4.4
iff I is a $\leq_s$-minimal stable trap space of $\mathsf{lfp}(P)$ by Corollary 4.2. □

Proposition 4.6. *Given two three-valued interpretations I_1 and I_2 of an NLP P, $I_1 \leq_u I_2$ iff $I_1^{\mathbf{u}} \subseteq I_2^{\mathbf{u}}$.*

Proof. We have $I_1 \leq_u I_2$
iff $I_1(v) \leq_u I_2(v)$ for all $v \in \mathsf{HB}_P$
iff either $I_1(v), I_2(v) \neq \star$ or $I_1(v) <_u I_2(v)$ for all $v \in \mathsf{HB}_P$
iff $I_1^{\mathbf{u}} \subseteq I_2^{\mathbf{u}}$. □

Corollary 4.5. *Given an NLP P, I is an L-stable model iff it is a $\leq_u$-minimal stable partial model.*

Proof. This immediately follows from Proposition 4.6. □

Proposition 4.7. *Consider an NLP P. For every supported trap space $I \in \mathsf{SuTS}(P)$, there is a supported partial model $J \in \mathsf{SuPM}(P)$ such that $J \leq_u I$.*

Proof. By Theorem 4.8, there is a unique supported partial model $J \in \mathsf{SuPM}(P)$ such that $J \leq_s I$. Since $I' \leq_s I$ implies $I' \leq_u I$, we can conclude the proof. Note that for $\leq_u$, such a supported partial model may not be unique. □

Corollary 4.6. *Consider an NLP P. A three-valued interpretation I is a $\leq_u$-minimal supported partial model of P iff I is a $\leq_u$-minimal supported trap space of P.*

Proof. We have $\mathsf{SuPM}(P) \subseteq \mathsf{SuTS}(P)$ by Proposition 4.2.

"$\Rightarrow$" Assume that I is not a $\leq_u$-minimal supported trap space of P. Then there exists $I' \in \mathsf{SuTS}(P)$ such that $I' <_u I$. By Proposition 4.7, there exists $J \in \mathsf{SuPM}(P)$ such that $J \leq_u I'$. This implies that $J <_u I$, which contradicts the $\leq_u$-minimality of I. Hence, I is a $\leq_u$-minimal supported trap space of P.

"$\Leftarrow$" This direction is trivial because $\mathsf{SuPM}(P) \subseteq \mathsf{SuTS}(P)$. □

Theorem 4.10. *Given an NLP P, I is an L-stable model iff I is a $\leq_u$-minimal stable trap space.*

Proof. We have P and $\mathsf{lfp}(P)$ have the same set of stable partial models by Theorem 4.6. They also have the same set of stable trap spaces by Corollary 4.3. Hence, it suffices to show that I is an L-stable model of $\mathsf{lfp}(P)$ iff I is a $\leq_u$-minimal stable trap space of $\mathsf{lfp}(P)$.

Recall that $\mathsf{lfp}(P)$ is a negative NLP. We have I is an L-stable model of $\mathsf{lfp}(P)$
iff I is a $\leq_u$-minimal stable partial model of $\mathsf{lfp}(P)$ by Corollary 4.5
iff I is a $\leq_u$-minimal supported partial model of $\mathsf{lfp}(P)$ by Theorem 4.5
iff I is a $\leq_u$ supported trap space of $\mathsf{lfp}(P)$ by Corollary 4.6
iff I is a $\leq_u$-minimal stable trap space of $\mathsf{lfp}(P)$ by Corollary 4.2. □

Proposition 4.8. *Given an NLP P, I is an L-stable model of P iff I is a $\leq_u$-minimal regular model.*

Proof. By Theorem 4.10, it suffices to show that I is a $\leq_u$-minimal stable trap space of P iff I is a $\leq_u$-minimal regular model of P. The forward direction is trivial because $\mathsf{RegM}(P) \subseteq \mathsf{StTS}(P)$, i.e., the set of regular models is a subset of the set of stable trap spaces. Now, we prove the backward direction.

Assume that I is a $\leq_u$-minimal regular model of P. Then I is a $\leq_u$-minimal regular model of $\mathsf{lfp}(P)$ by Theorem 4.6. Suppose that I is not a $\leq_u$-minimal stable trap space of $\mathsf{lfp}(P)$. Then there is a stable trap space I' of $\mathsf{lfp}(P)$ such that $I' <_u I$. By Theorem 4.9, the regular models of $\mathsf{lfp}(P)$ are the $\leq_s$-minimal

stable trap spaces of $\mathsf{lfp}(P)$. This implies that there is a regular model J of $\mathsf{lfp}(P)$ such that $J \leq_s I'$. Since $J \leq_s I'$ implies $J \leq_u I'$, $J <_u I$. By Theorem 4.6, J is a regular model of P, which contradicts the $\leq_u$-minimality of I. Hence, I is a $\leq_u$-minimal stable trap space of $\mathsf{lfp}(P)$. By Corollary 4.3, I is a $\leq_u$-minimal stable trap space of P. □

Corollary 4.7. *If an NLP P has no L-stable model, then it has infinitely many regular models.*

Proof. By Proposition 4.8, the set of L-stable models of P coincides with the set of $\leq_u$-minimal regular models of P. If $\mathsf{RegM}(P)$ is a finite set, then it always has a $\leq_u$-minimal element, i.e., the set of $\leq_u$-minimal regular models of P is non-empty, leading to P has an L-stable model, which is a contradiction. Hence, P has infinitely many regular models. □

Theorem 5.2. *An NLP P always has a $\subseteq$-minimal stable (or supported) trap set.*

Proof. For the case of $\subseteq$-minimal stable trap sets, this immediately follows from Lemma 4.1 and the fact that P has at least one stable trap set by Proposition 3.1.

For the case of $\subseteq$-minimal supported trap sets, this immediately follows from Lemma 4.2 and the fact that P has at least one supported trap set by Proposition 3.1. □

Alternative proof of Theorem 5.1. By Theorem 5.2, P has a $\subseteq$-minimal stable trap set S. We show that S is a stable class.

By definition, $\{F_P(I) \mid I \in S\} \subseteq S$. Suppose that $\{F_P(I) \mid I \in S\} \subsetneq S$. Then there exists $J \in S$ such that for all $I \in S$, $F_P(I) \neq J$. We have $\{F_P(I) \mid I \in S \setminus \{J\}\} \subseteq \{F_P(I) \mid I \in S\} \subseteq S \setminus \{J\}$. Thus $S \setminus \{J\}$ is a supported trap set of P, which contradicts the $\subseteq$-minimality of S. Hence $\{F_P(I) \mid I \in S\} = S$, i.e., S is a supported class of P. □

Theorem 5.3. *Any NLP P has a supported class.*

Proof. By Theorem 5.2, P has a $\subseteq$-minimal supported trap set S. We show that S is a supported class.

By definition, $\{T_P(I) \mid I \in S\} \subseteq S$. Suppose that $\{T_P(I) \mid I \in S\} \subsetneq S$. Then there exists $J \in S$ such that for all $I \in S$, $T_P(I) \neq J$. We have $\{T_P(I) \mid I \in S \setminus \{J\}\} \subseteq \{T_P(I) \mid I \in S\} \subseteq S \setminus \{J\}$. Thus $S \setminus \{J\}$ is a supported trap set of P, which contradicts the $\subseteq$-minimality of S. Hence $\{T_P(I) \mid I \in S\} = S$, i.e., S is a supported class of P. □

Corollary 5.1 (main result). *An NLP P always has a strict stable (or supported) class.*

Proof. This immediately follows from Theorem 5.2 and Theorem 4.1 (resp. Theorem 4.2). □

Corollary 5.2. *An NLP P always has a regular model.*

Proof. This immediately follows from Theorem 4.9 and Theorem 3.1. □

B Revisit of the Characterizations of Strict Stable and Supported Classes

We first revisit the basic definitions and results of [27] on stable and supported classes.

Definition B.1 ([27])**.** *Consider an NLP P and a two-valued interpretation I of P. A sequence of applications of an operator on the set of two-valued interpretations of P is called an* orbit*. The orbit of I w.r.t. T_P is the sequence $\langle T_P^k(I)\rangle_{k\in\omega}$, where $T_P^0(I) = I$ and $T_P^{k+1}(I) = T_P(T_P^k(I))$. Similarly, we have the definition for the orbit of I w.r.t. F_P.*

Definition B.2 ([27])**.** *Consider an NLP P and a two-valued interpretation I of P. Let $\mathscr{T}_P(I)$ be the set of two-valued interpretations defined as:*

$$\mathscr{T}_P(I) = \{T_P^k(I) \mid k \in \omega\}.$$

Let $\mathscr{F}_P(I)$ be the set of two-valued interpretations defined as:

$$\mathscr{F}_P(I) = \{F_P^k(I) \mid k \in \omega\}.$$

Theorem B.1 ([27])**.** *Given an NLP P, a non-empty set S of two-valued interpretations is a strict stable (resp. supported) class of P iff $\mathscr{T}_P(I) = S$ (resp. $\mathscr{F}_P(I) = S$) for every $I \in S$.*

Theorem B.1 provides an alternative characterization for strict stable (or supported) classes. This theorem mainly relies on the definitions of stable and supported orbits (see Definition B.1). This may be a problem here, as these definitions only limit the length of the sequence to ω (the first infinite limit ordinal). Since the sequence can traverse through the set of all two-valued interpretations, which may be an infinite set, thus the length can be a bigger ordinal. With Definition B.1, it seems that Theorem B.1 only consider Datalog$^\neg$ programs rather than any NLPs. We fix the above problem as follows.

First, we recursively define the stable or supported reachable closure of a two-valued interpretation over all ordinals.

Definition B.3. *Given an NLP P, let I is a two-valued interpretation of P. The* supported reachable closure *of I is defined as:*

$$\begin{aligned}\Pi_P^I \uparrow 0 &= \{I\}\\ \Pi_P^I \uparrow \alpha+1 &= \Pi_P^I \uparrow \alpha \cup \bigcup_{J\in\Pi_P^I\uparrow\alpha} \{T_P(J)\}.\end{aligned}$$

If β is a limit ordinal, then by construction $\{\Pi_P^I \uparrow \alpha \colon \alpha < \beta\}$ is a $\subseteq$ chain. Define $\Pi_P^I \uparrow \beta = sup_\subseteq\{\Pi_P^I \uparrow \alpha \colon \alpha < \beta\}$, where $sup_\subseteq$ denotes the least upper bound w.r.t. $\subseteq$. By replacing $T_P(J)$ and Π_P^I with $F_P(J)$ and Σ_P^I respectively, we obtain the definition of the stable reachable closure *of I.*

Then the two following propositions show that the stable or supported reachable closure eventually reaches a constant w.r.t. any two-valued interpretation. The key to their proofs is the use of Hartogs' lemma (see Lemma 2.3).

Proposition B.1. *Consider an NLP P. For any two-valued interpretation I of P, there is the unique least ordinal α_I such that $\Pi_P^I \uparrow \alpha_I + 1 = \Pi_P^I \uparrow \alpha_I$.*

Proof. We have $\Pi_P^I \uparrow$ is an increasing function from the ordinals into 2^{HB_P}. It cannot be strictly increasing, as if it was, then we would have an injection from the ordinals into 2^{HB_P}, which violates Lemma 2.3. Hence, $\Pi_P^I \uparrow$ must be eventually constant. It follows that for some ordinal α, $\Pi_P^I \uparrow \alpha + 1 = \Pi_P^I \uparrow \alpha$. Let α_I be the least ordinal such that $\Pi_P^I \uparrow \alpha_I + 1 = \Pi_P^I \uparrow \alpha_I$. Since the ordinals are ordered, α_I is unique. □

Proposition B.2. *Consider an NLP P. For any two-valued interpretation I of P, there is the unique least ordinal α_I such that $\Sigma_P^I \uparrow \alpha_I + 1 = \Sigma_P^I \uparrow \alpha_I$.*

Proof. Replace Π_P^I with Σ_P^I in the proof of Proposition B.1. □

By using transfinite induction (see more details in [30]), we prove that the supported reachable closure is closed under the supported trap set property.

Lemma B.1. *Consider an NLP P. Let S be a supported trap set of P. Then for all $I \in S$, $\Pi_P^I \uparrow \alpha \subseteq S$ for all ordinal α.*

Proof. We prove this by using transfinite induction.

Base case: $\Pi_P^I \uparrow 0 = \{I\} \subseteq S$.

Successor case: $\Pi_P^I \uparrow \alpha + 1 = \Pi_P^I \uparrow \alpha \cup \bigcup_{J \in \Pi_P^I \uparrow \alpha} \{T_P(J)\} \subseteq S \cup \bigcup_{J \in S} \{T_P(J)\}$ because $\Pi_P^I \uparrow \alpha \subseteq S$ by the induction hypothesis. Then $\Pi_P^I \uparrow \alpha + 1 \subseteq S \cup S = S$ because S is a supported trap set.

Limit case: $\Pi_P^I \uparrow \alpha = \sup_{\subseteq} \{\Pi_P^I \uparrow \beta : \beta < \alpha\} \subseteq S$ because $\Pi_P^I \uparrow \beta \subseteq S$ for all $\beta < \alpha$ by the induction hypothesis. □

On the other hand, we prove that the constant that the supported reachable closure reaches is a supported trap set w.r.t. any two-valued interpretation.

Lemma B.2. *Given an NLP P, for any two-valued interpretation I of P, $\Pi_P^I \uparrow \alpha_I$ is a supported trap set of P.*

Proof. By definition of Π_P^I, we have $\Pi_P^I \uparrow \alpha_I + 1 = \Pi_P^I \uparrow \alpha_I \cup \bigcup_{J \in \Pi_P^I \uparrow \alpha_I} \{T_P(J)\}$. By definition of α_I, we have $\Pi_P^I \uparrow \alpha_I + 1 = \Pi_P^I \uparrow \alpha_I$. Set $S = \Pi_P^I \uparrow \alpha_I$. It follows that $S = S \cup \bigcup_{J \in S} \{T_P(J)\}$. Then $\bigcup_{J \in S} \{T_P(J)\} \subseteq S$, implying that S is a supported trap set of P. □

Combining Lemma B.1 and Lemma B.2, we now obtain a new characterization for $\subseteq$-minimal supported trap sets of an NLP:

Theorem B.2. *Given an NLP P, S is a $\subseteq$-minimal supported trap set of P iff $\Pi_P^I \uparrow \alpha_I = S$ for all $I \in S$.*

Proof. "$\Rightarrow$" Assume that S is a $\subseteq$-minimal supported trap set of P. Suppose that there exists $I \in S$ such that $\Pi_P^I \uparrow \alpha_I \neq S$. By Lemma B.1, $\Pi_P^I \uparrow \alpha_I \subseteq S$ since S is a supported trap set. Then $\Pi_P^I \uparrow \alpha_I \subset S$. By Lemma B.2, $\Pi_P^I \uparrow \alpha_I$ is a supported trap set, which contradicts the $\subseteq$-minimality of S. Hence, $\Pi_P^I \uparrow \alpha_I = S$ for all $I \in S$.

"$\Leftarrow$" Assume that $\Pi_P^I \uparrow \alpha_I = S$ for all $I \in S$. We have $\Pi_P^I \uparrow \alpha_I + 1 = S = S \cup \bigcup_{J \in S} \{T_P(J)\}$. This implies that $\bigcup_{J \in S} \{T_P(J)\} \subseteq S$, thus S is a supported trap set of P. Suppose that S is not $\subseteq$-minimal. Then there is a supported trap set S' of P such that $S' \subset S$. Consider $J \in S'$. By Lemma B.1, $\Pi_P^I \uparrow \alpha_J \subseteq S' \subset S$, which is a contradiction because $J \in S$. Hence, S is a $\subseteq$-minimal supported trap set of P. □

This is similar for stable trap sets by replacing Π_P^I by Σ_P^I in the proof of Theorem B.2:

Theorem B.3. *Given an NLP P, S is a $\subseteq$-minimal stable trap set of P iff $\Sigma_P^I \uparrow \alpha_I = S$ for all $I \in S$.*

Proof. Replace Π_P^I by Σ_P^I in the proof of Theorem B.2. □

Theorem B.2 and Theorem B.3 are the corrected version of Theorem B.1 in the case of NLPs, as we show that the $\subseteq$-minimal supported (resp. stable) trap sets are equivalent to strict supported (resp. stable) classes (see Theorem 4.1 and Theorem 4.2). In the case of Datalog$^\neg$ programs, HB_P is finite, then $\mathscr{T}_P(I) = \Pi_P^I \uparrow \omega$ (resp. $\mathscr{F}_P(I) = \Sigma_P^I \uparrow \omega$). Hence, in this case, Theorem B.2 and Theorem B.3 are equivalent to Theorem B.1.

C Revisit of the Existence of L-stable Models

The existence of L-stable models in an NLP is stated and proved in [35, Proposition 5.11]. The main idea of the proof is similar to what is used for proving the existence of M-stable models in an NLP (see [35, Theorem 5.3]). However, this may pose a problem here because the case of L-stable models is actually not the same as the case of M-stable models. More specifically, the union of all the partial stable models in a maximal chain w.r.t. $\subseteq$-minimal $I^{\mathbf{u}}$ might be not a partial stable model (even the union does not exist, see Example C.1), whereas the union of all the stable partial models in a $\leq_i$-chain is a stable partial model as shown in the proof of [35, Theorem 5.3]. This problem is more justified by the bijection between the L-stable models of an NLP and the semi-stable extensions of an Abstract Argumentation Framework (AF) [9]. It has been shown that there is an infinite AF without a semi-stable extension [6].

Example C.1. *Consider the NLP $P = \{a \leftarrow \sim b; b \leftarrow \sim a; c \leftarrow \sim c\}$. The program P has three stable partial models: $I_1 = \{a = \star, b = \star, c = \star\}$, $I_2 = \{a = 0, b = 1, c = \star\}$, and $I_3 = \{a = 1, b = 0, c = \star\}$. Using the notations of [35], we have $I_1 = \langle \emptyset, \emptyset \rangle$, $I_1 = \langle \{b\}, \{a\} \rangle$, and $I_1 = \langle \{a\}, \{b\} \rangle$. Every maximal chain w.r.t. $\subseteq$-minimal $I^{\mathbf{u}}$ of P contains all I_1, I_2, and I_3. However, the union of all elements in such a chain is $\langle \{a,b\}, \{a,b\} \rangle$, which does not represent a valid three-valued interpretation. Hence, the union does not exist.*

We make the following two conjectures, inspired by existing related results in abstract argumentation frameworks [6, 38].

Conjecture C.1. *An internal-variable-free NLP P has at least one L-stable model.*

Conjecture C.2. *There exists an NLP P such that P has no L-stable model.*

We here prove a special case of Conjecture C.1, however this conjecture and Conjecture C.2 are still open. This result is meaningful because 1) the class of uni-rule NLPs is easily identifiable by a syntatic characterization, 2) uni-rule Datalog$^\neg$ programs are as hard as general Datalog$^\neg$ programs w.r.t the stable model semantics, since identifying whether a uni-rule Datalog$^\neg$ program has a stable model or not is NP-complete [37], and 3) this class is straightforwardly related to (possibly infinite) AFs [17, 11].

Theorem C.1. *Consider an NLP P. If P is uni-rule, then it has at least one L-stable model.*

Proof. Since P is uni-rule, it is easy to derive that $\mathsf{lfp}(P)$ is also uni-rule thanks to the definition of the least fixpoint. If $\mathsf{lfp}(P) = \emptyset$, then it trivially has a L-stable model I^* where $I^*(a) = 0$ for every $a \in \mathsf{HB}_{\mathsf{lfp}(P)}$. We then only consider the case $\mathsf{lfp}(P) \neq \emptyset$. If there is an atom $a \in \mathsf{HB}_{\mathsf{lfp}(P)}$ not being the head of any rule in $\mathsf{lfp}(P)$, it is always 0 in any stable or supported partial model of $\mathsf{lfp}(P)$. Hence we can "propagate" such atoms to get a new NLP P' such that the set of stable partial models of P' one-to-one corresponds to the set of stable partial models of $\mathsf{lfp}(P)$. This implies that the set of L-stable models of P' one-to-one corresponds to the set of L-stable models of $\mathsf{lfp}(P)$. Since $\mathsf{lfp}(P)$ is uni-rule and negative, P' also does. There is a (possibly infinite) AF (say $\mathscr{A}$) corresponding to P' where the set of arguments of $\mathscr{A}$ is $\mathsf{HB}_{P'}$ and each atom in the body of a rule whose head is a corresponds to an attacker of a in $\mathscr{A}$. Since the body of a rule is finite by definition, $\mathscr{A}$ is finitary, i.e., every argument has finitely many attackers [6].

A labelling of an AF is identical to a three-valued interpretation of an NLP. An extension of an AF is defined as a subset of arguments and correponds to a two-valued interpretation of an NLP. There is a one-to-one correspondence between the set of labellings and the set of extensions of a given type [11]. [6, Theorem 14] points out that a finitary AF has at least one semi-stable extension. We can easily to derive the existence of semi-stable labellings in $\mathscr{A}$.

A complete labelling I is a semi-stable labelling if $I^{\mathbf{u}}$ is $\subseteq$-minimal [9]. By the definition of complete labellings [11], we can derive that the set of complete lablellings of $\mathscr{A}$ coincides with the set of supported partial models of P'. Since the stable partial models of P' coincide with the supported partial models of P' by Theorem 4.5, P' has at least one L-stable model, leading to $\mathsf{lfp}(P)$ does. Since P and $\mathsf{lfp}(P)$ have the same set of stable partial models by Theorem 4.6, P has at least one L-stable model. □

Reasoning about Medical Triage Optimization with Logic Programming

Jaikrishna Manojkumar Patil
Syracuse University
Syracuse, NY, USA
jpatil01@syr.edu

Adam Chapman
SimWerx
Denver, CO, USA
adam@simwerx.com

Richard Knuszka
SimWerx
Denver, CO, USA
rich@simwerx.com

John Chapman
SimWerx
Denver, CO, USA
john@simwerx.com

Paulo Shakarian
Syracuse University
Syracuse, NY, USA
pashakar@syr.edu

We present a logic programming framework that orchestrates multiple variants of an optimization problem and reasons about their results to support high-stakes medical decision-making. The logic programming layer coordinates the construction and evaluation of multiple optimization formulations, translating solutions into logical facts that support further symbolic reasoning and ensure efficient resource allocation—specifically targeting the "right patient, right platform, right escort, right time, right destination" principle. This capability is integrated into GuardianTwin, a decision support system for Forward Medical Evacuation (MEDEVAC), where rapid and explainable resource allocation is critical. Through a series of experiments, our framework demonstrates an average reduction in casualties by 35.75% compared to standard baselines. Additionally, we explore how users engage with the system via an intuitive interface that delivers explainable insights, ultimately enhancing decision-making in critical situations. This work demonstrates how logic programming can serve as a foundation for modular, interpretable, and operationally effective optimization in mission-critical domains.

1 Introduction

Traditional medical triage algorithms, such as the Glasgow Coma Scale (GCS), the New Injury Severity Score (NISS), the Revised Trauma Score (RTS), and the LIFE score, efficiently categorize and prioritize patient care based on factors like injury severity, mental status, and vital signs [21, 15, 9, 19]. However, while these medical triage algorithms help efficiently categorize and prioritize patient care near the point of injury or at casualty collection points, these algorithms do not account for the strategic and logistical challenges critical to Forward Medical Evacuation (MEDEVAC) [3] operations, particularly in high-intensity military conflicts like the 2022 Russo-Ukraine war. Current MEDEVAC mission planning remains largely manual, making it difficult to scale in complex operational environments. Effective mission planning must consider factors such as geographic location, asset availability, proximity to care facilities, environmental conditions, and real-time operational intelligence. Given these challenges, the principle of "right patient, right platform, right escort, right time, right destination" becomes paramount [3]. For example, a single patient triaged as "Immediate" might consume resources that could be better distributed to save multiple personnel categorized as "Urgent." This underscores the need for a system that optimizes resource allocation and adjusts dynamically to operational changes. While optimization techniques are well-suited for addressing these challenges, there are questions about which type of optimization criteria to select [2, 11], how to formulate the problem for a given scenario, and how to

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 320–333, doi:10.4204/EPTCS.439.22

© Patil, Chapman, Knuszka, Chapman & Shakarian
This work is licensed under the
Creative Commons Attribution License.

decide on a proper allocation of resources given a choice of solvable optimization problems. These gaps motivate the use of logic programming, which enables symbolic reasoning over constraints, preferences, and multiple optimization formulations in a structured and explainable manner.

In this paper, we introduce a logic programming based framework that allows for the *orchestration* of one or more optimization problems for medical triage. By creating a logical language to not only model medical triage situations but also to frame and reason about the solutions to optimization problems, we can create various optimization problems on the fly with the logic, send the results to an external solver, and then convert the results back into logical facts that allows for a form of "what-if" reasoning about the outcome if resource decisions are made in a certain manner. In our paper, we describe how such reasoning can be done either via displaying results back to the user, or in automated fashion by reasoning over optimization problem outcome(s). To operationalize this framework in real-world mission planning, we extend our GuardianTwin triage platform, which integrates real-time data and decision-making algorithms to support Forward MEDEVAC coordination. GuardianTwin converts optimization outputs into intuitive interface elements, helping field medics and mission planners make informed decisions under pressure. We then provide a suite of results where we examine the performance of the approach for cases where single and multiple optimization problems are formulated and show this provides on average at least 35.75% reduction in casualties over standard baselines. We also show how our approach can reason over multiple optimization problems as well as how it can be employed in time-evolving situations. While our primary focus in this paper is on Forward MEDEVAC operations, the logic programming framework we present is generalizable to other domains that require dynamic resource prioritization under constraints. The core idea of using logic rules to orchestrate optimization problem generation, reasoning over solutions, and adapting plans over time applies to any setting where explainability and constraint handling are essential. As such, our framework offers a reusable and interpretable foundation for decision support in time-critical domains beyond the military context.

2 Background

In this section we review patient criticality scoring (a feature included in the current version of GuardianTwin) and describe current thinking about medical triage with resource constraints.

Patient Criticality Scoring. Current triage algorithms, such as NISS, RTS, and LIFE, have evolved to provide quantitative assessments of patient severity based on vital signs, injury scales, and other critical factors. NISS ranges from 0 (minor injuries) to 75 (severe injuries) and is based on the Abbreviated Injury Scale(AIS) for body regions, considering multiple injuries in a casualty. RTS ranges from 0 (most severe) to 12 (least severe) and incorporates vital signs such as systolic blood pressure, respiratory rate, and GCS. The LIFE score, ranging from 0 (most severe) to 100 (least severe), combines both AIS and vital information to provide a comprehensive severity score. In GuardianTwin and in this work, we have developed a simplified LIFE scoring system that equally weighs NISS and RTS scores to enhance decision-making in complex scenarios. Furthermore, to integrate multiple ranking systems, we normalize each score within a range of 0 (least priority) to 1 (highest priority). This normalization allows for consistent comparison across different scoring systems, accommodating situations where complete information might not be available for every casualty. In this paper, we will use the notation $life$, rts, $niss$ to denote functions that accept a vector representation of a patient and return a normalized score as described above.

Optimizing Triage with real world constraints. To illustrate the challenges in MEDEVAC mission planning, we present a scenario (see Figure 1(left)) with 5 assets and 11 casualties dispersed over a large

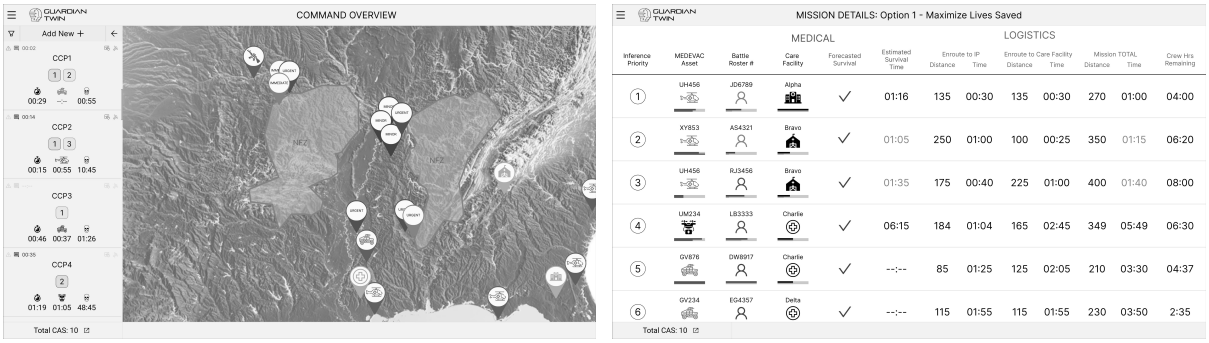


Figure 1: Left: GuardianTwin Command view. Right: GuardianTwin Mission Details view (prototype).

geographical area: 3 "Immediate," 5 "Urgent," and 3 "Minor." If asset allocation follows these categories, 3 "Immediate" and 2 "Urgent" casualties would receive resources, but this approach overlooks factors like response time, airspace control, and logistical considerations, potentially resulting in additional casualties. Given these complexities, framing mass casualty triage as an optimization problem becomes essential. However, the criteria for optimization depend on the specific situation. *Urgency-based triage* prioritizes immediate intervention. Meanwhile *reverse triage* focuses on return-to-duty time and *situational triage* integrates operational constraints [2, 11].

Furthermore in MEDEVAC operations, [10] optimized Mobile Aeromedical Staging Facilities (MASFs) deployment and helicopter allocation based on logistical factors. They also focus on logistical considerations like coverage of MASF and capacity of helicopters. Similarly, OPTEVAC model [20] uses linear programming model to determine the optimal quantity and placement of evacuation assets based on the factors like available assets, area of battlefield, and preferred locations of medical care facilities. However, these models lack the framework to orchestrate multiple optimization problems and provide explainable traces that would help for taking next steps. Our logic-based framework bridges this gap, enabling decision support by integrating factors like distance to care facilities, airspace control, and MEDEVAC asset availability.

Overview of GuardianTwin System. GuardianTwin is a decision support platform designed mainly to optimize the survivability and resource management of battlefield casualties, particularly in large-scale combat operations (LSCO). GuardianTwin integrates real-time data, predictive modeling, and logic programming to maximize casualty survival and streamline resource management. The system creates virtual models of real-world entities such as casualties, assets, and care facilities. Due to the sensitive nature of this domain, it is not possible to get real-world data. However, GuardianTwin generates exercise-grade datasets specifically engineered for military exercises. These datasets are designed to reflect realistic casualty models, operational constraints, and logistical challenges, ensuring the relevance of platform to real-world applications. This data has been developed independently by GuardianTwin framework and is not a contribution of this work. GuardianTwin serves three primary end-user groups: field medics at the Casualty Collection Point (CCP), Personnel Recovery Coordination Cell (PRCC) duty officers, and Patient Evacuation Coordination Cell (PECC) duty officers. It offers real-time triage support, predictive analytics, and resource prioritization to enhance decision-making in critical conditions. PRCC and PECC officers use the system to manage personnel recovery and evacuation planning, respectively.

Related Work. Various approaches use reinforcement learning (RL) to optimize medical triage operations [22, 14] including in military medical triage [6, 17]. Similar RL approach is also used in human-

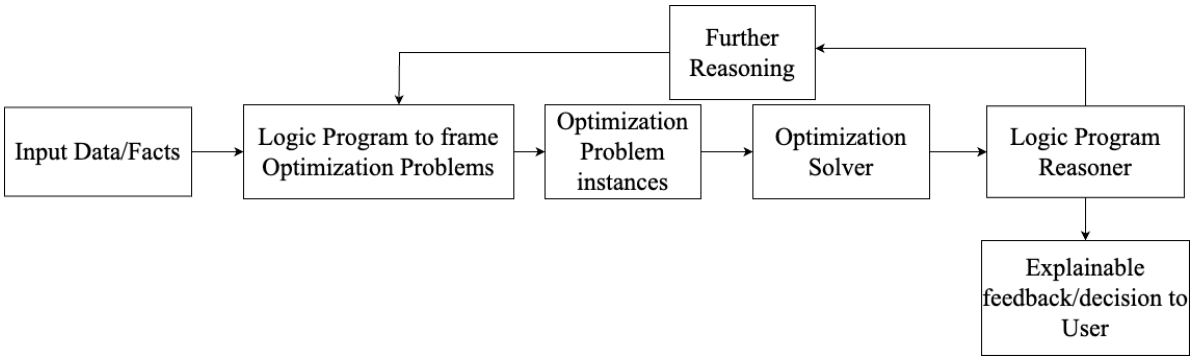


Figure 2: Overview of the framework for logic based orchestration of medical triage optimization.

itarian logistics for resource allocation [23]. However, these approaches consider optimization criteria established ahead of time, which does not enable the user to understand the trade-off of various criteria (nor establish their own criteria for such trade offs in a logic program). Other approaches include various machine learning models [16, 4, 5] which like the reinforcement learning approach all consider a single optimization criteria. Medical triage is also studied in the field of AI for issues concerning recognition of casualties for robotic systems [13], and fairness in the triage process [7]. These issues do not consider the automated formulation of the triage problem as we do, but at the same time represent important concerns that would be worth considering in extensions to the framework described in this paper.

3 Triage Logic and UI Prototypes

In this section we describe how we use logic programming to orchestrate one or more medical triage optimization problems as well as show examples of how users interact with the logic program via the GuardianTwin user interface. We show an overall diagram of this approach in Figure 2. The key idea is that initial facts (including facts concerning the setup of the logic program) are input into the system, resulting in a deductive process that produces one or more optimization problems that are solved by an external solver. The results of the solver are then re-interpreted as facts and added back into the logic program, allowing for further automated reasoning and/or analysis by the user.

Logical Formalism. In this section, we briefly describe our logical language used for describing casualty triage based on a first order logic using datalog-style rules [1, 18]. We assume a set of constant, variable, and predicate symbols (resp., $\mathscr{C}$, $\mathscr{V}$, $\mathscr{P}$) where the set of constants is divided into domains (denoted with the symbol $\mathscr{D}$ often with a subscript). Three key domains of importance we shall denote $\mathscr{D}_{\mathbf{p}}, \mathscr{D}_{\mathbf{a}}, \mathscr{D}_{\mathbf{f}}$ for the sets of constants representing the sets of constants associated with geolocated casualties (personnel), assets, and care facilities respectively and are understood in the context of the MEDEVAC problem set. Two other sets of constants, $\mathscr{D}_{\mathbf{o}}, \mathscr{D}_{\mathbf{c}}$ are associated with optimization criteria and constraints respectively. These constants are used for allowing the reasoning system to formulate an optimization problem.

Additionally, we denote the set of predicates as $\mathscr{P}$ and such predicates with terms (constants or variables) form atoms in the usual way. For sake of brevity, we will only mention a few specific ones here. There are three resulting from our overall process that describe the output - and actually are used in facts created by optimization problems orchestrated by the reasoning process. First the unary predicate `evac` denotes if a casualty has been evacuated (e.g. $\texttt{evac}(p)$ means that p was evacuated). Next, the binary predicates $\texttt{assign}_{\mathbf{a}}, \texttt{assign}_{\mathbf{f}}$ take a casualty as the first argument and an asset (resp. facility)

as the second - so $\texttt{assign}_{\mathbf{a}}(p,a)$ means that casualty p was assigned asset a and $\texttt{assign}_{\mathbf{f}}(p,f)$ means casualty p was assigned to facility f. For the reasoning process that stages the optimization problem, there is the binary predicate `score` that accepts an element from $\mathscr{D}_{\mathbf{p}}$ as the first argument and a number in $(0,1)$ as the second. In our framework, we allow this predicate to only appear in rule heads, as it is used to determine the type of score assigned to a casualty based on available data. There are two unary predicates that are also used to help stage the optimization problem. They are $\texttt{use}_{\mathbf{c}}$, $\texttt{use}_{\mathbf{o}}$ and take constants from $\mathscr{D}_{\mathbf{c}}, \mathscr{D}_{\mathbf{o}}$ respectively. Here, $\texttt{use}_{\mathbf{c}}(c)$ means that we must enforce constraint c during an optimization procedure and $\texttt{use}_{\mathbf{o}}(o)$ means that we must use optimization criteria o. Note that for only one $o \in \mathscr{D}_{\mathbf{o}}$ can $\texttt{use}_{\mathbf{o}}(o)$ be true when using a single optimization problem. If we are using the framework to explore multiple optimization problems (as described later in the experiments), we extend the aforementioned predicates to binary predicates (we omit the details here for brevity).

We note that predicates like $\texttt{use}_{\mathbf{c}}, \texttt{use}_{\mathbf{o}}$ are a direct result of the initial deductive process (which results in the setup of one or more optimization problems), while atoms formed with predicates `evac`, $\texttt{assign}_{\mathbf{a}}$, $\texttt{assign}_{\mathbf{f}}$ are actually created as facts after one or more optimization problems are solved (for the case where we have more than one problem, we extend these predicates with an additional argument, a natural number i indicating the ith optimization problem result). For the results of a single optimization problem, we have created a "Mission Details" view screen for GuardianTwin (see Figure 1(right))summarizing which atoms formed with `evac`, $\texttt{assign}_{\mathbf{a}}$, $\texttt{assign}_{\mathbf{f}}$ have resulted from a given optimization problem.

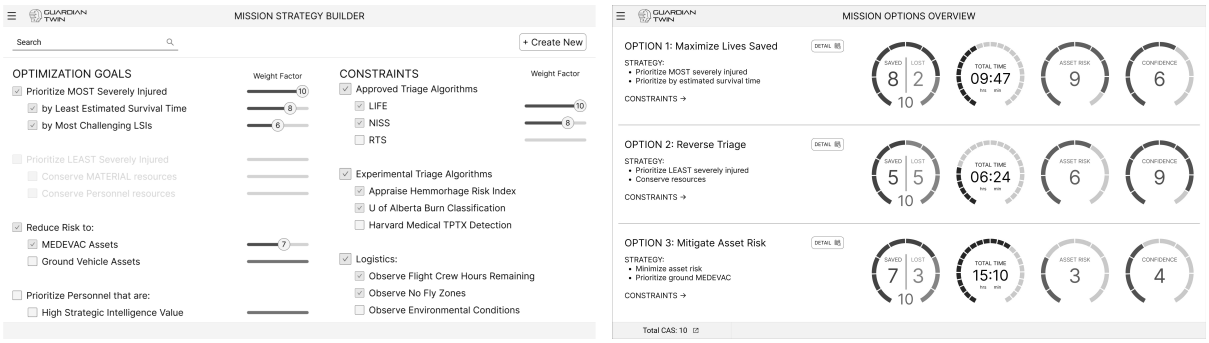


Figure 3: Left: GuardianTwin Strategy Builder view. Right: Example results for three different optimization problems viewed in a prototype GuardianTwin interface.

We will use strong negation ($\neg$) to denote a given atom is false (as opposed to merely "not true") - as our underlying logic allows uncertainty as a default truth value. We will use the term "facts" to refer to a set of atoms or negations thought to be initially true. We will also use atoms and negations in logic program rules (formal syntax is described in [18]). A *logic program,* often denoted as Π in this paper is simply a set of facts and rules. Unless mentioned explicitly, we shall assume that this logic program is fixed. Table 1 provides an example with the first rule states if *insult severity scores* are *available* for the casualty and *vitals* is *not available*, then we will use NISS scoring to compute triage score. On the other hand, second rule states if *insult severity scores* are *not available* for the casualty and *vitals* is *available*, then we will use RTS scoring to compute triage score. Third rule states that we will use LIFE scoring if *both* insult severity score and vitals are *available*.

For deductive inference with logic program Π, several different paradigms are possible. We refer the reader to references such as [12, 18, 1] for more complete review of syntax and semantics we use in this work. We selected this particular variant of logic programming as it allows for the representation of

Table 1: Logic program for framing optimization problems based on different criteria

$\texttt{score}(P, niss(P)) \leftarrow \texttt{insultsAvailable}(P), \neg\texttt{vitalsAvailable}(P)$
$\texttt{score}(P, rts(P)) \leftarrow \neg\texttt{insultsAvailable}(P), \texttt{vitalsAvailable}(P)$
$\texttt{score}(P, life(P)) \leftarrow \texttt{insultsAvailable}(P), \texttt{vitalsAvailable}(P)$
$\texttt{use}(o_{rtd}) \leftarrow$

uncertainty. Also, as per the aforementioned work, we assume initial values of all atoms or negations to be "uncertain" if not specified in an initial set of facts. We assume that the logic is monotonic in that the truth value of ground atoms starts out as uncertain and moves to either true or false - but cannot change further (attempts to do so will result in an inconsistency as per [18]). We will assume that the logic program in this work is consistent (and note in the above framework, the results of [18] tell us we can efficiently identify inconsistencies). As a result our simple logic is amenable to a fixpoint-based deduction process (see the aforementioned references for details) that provides a set of all atoms and negations that can be derived from program Π (either in the initial set of facts or derived from rules). This process can be done efficiently as per [12, 18] and implemented in PyReason [1] so we omit the details here. For our purposes we care about the result of the deduction process. As such is the case, for a given logic program Π, we will use the notation $\Gamma^*(\Pi)$ (a slight abuse of fixpoint notation used in [12]) to refer to the set of all facts returned by the logic program. Note that for the initial deductive process the key result of $\Gamma^*(\Pi)$ is to frame the optimization problems and perform subsequent reasoning.

Using Logic Programming to Orchestrate Optimization Problems. Now we describe how we use this logical language to orchestrate optimization problems. For starters, we define several functions over our set of constants. In the background we already mentioned $life$, rts, and $niss$ which map $\mathscr{D}_{\mathbf{p}}$ to $(0,1)$. Based on these, we define function $scr : \mathscr{D}_{\mathbf{p}} \rightarrow (0,1)$ defined as follows:

$$scr(p) = \sup\{x \textit{ such that } \texttt{score}(p,x) \in \Gamma^*(\Pi)\}$$

We note that this function depends on `score` atoms deduced by Π (recall that Π is fixed and `score` only appears in rule heads). The intuition here is that we now obtain a single scalar individual triage score for each casualty - and this can be derived from multiple rules (e.g., those in Table 1). We note that unlike imputation, as used in traditional data mining, this approach allows for the logic around the triage score to reside in the logic program that is also used to formulate the optimization problem - which allows for more streamline integration with the user interface (e.g., the "Strategy Builder" depicted in Figure 3 (left)) as well as allow better maintainability of the GuardianTwin codebase. Other functions concerning return-to-duty time, and time required for life-saving intervention for a given casualty are denoted by $rtd : \mathscr{D}_{\mathbf{p}} \rightarrow \Re^+$ and $lsi : \mathscr{D}_{\mathbf{p}} \rightarrow \Re^+$. These are defined in a similar fashion. Additionally, the function $trip : \mathscr{D}_{\mathbf{p}} \times \mathscr{D}_{\mathbf{a}} \times \mathscr{D}_{\mathbf{f}} \rightarrow \Re^+$, for some arguments p, a, f (i.e., $trip(p,a,f)$ is the total trip time for casualty p to facility f via asset a. Likewise functions $trip_{\mathbf{p}} : \mathscr{D}_{\mathbf{a}} \times \mathscr{D}_{\mathbf{p}} \rightarrow \Re^+$ and $trip_{\mathbf{f}} : \mathscr{D}_{\mathbf{p}} \times \mathscr{D}_{\mathbf{f}} \rightarrow \Re^+$ represent travel times from the asset to casualty and casualty to care facility. In orchestrating medical triage optimization, it is useful to specify constraints of different types (and, in fact these constraints are associated with elements of $\mathscr{D}_{\mathbf{c}}$. These constraints are applied on subsets of ground atoms *after* the

completion of the deduction process and we list several we use in this paper below.

$$\begin{aligned}
c_{lsi} :=&\quad \forall \texttt{assign}_{\mathbf{a}}(p,a) \in \Gamma^*(\Pi): \quad trip_{\mathbf{p}}(a,p) \leq lsi(p)\\
c^k_{scr} :=&\quad \forall \texttt{evac}(p) \notin \Gamma^*(\Pi): \quad scr(p) \leq k\\
c^k_{rtd} :=&\quad \forall \texttt{evac}(p) \notin \Gamma^*(\Pi): \quad rtd(p) \geq k\\
c^k_{air} :=&\quad \forall \texttt{assign}_{\mathbf{a}}(p,a) \in \Gamma^*(\Pi): \quad trip_{\mathbf{p}}(a,p) \leq k \textit{ and}\\
&\quad \forall \texttt{assign}_{\mathbf{f}}(p,f) \in \Gamma^*(\Pi): \quad trip_{\mathbf{f}}(p,f) \leq k
\end{aligned}$$

Intuitively, constraint c_{lsi} states that any casualty assigned an asset is provided that asset before life saving intervention is required. Constraint c^k_{scr} (defined for some constant k) ensures that no casualty not evacuated has a triage score greater than k (and c^k_{rtd} is simply the analog for return to duty). Constraint c^k_{air} (defined for some constant k) ensures that the time in the air for an asset on any leg of the journey does not exceed k time units. We note that not all of these constraints are necessarily to be enforced at once (and we may even have different constraints used for different optimization problems orchestrated by the logic program). The key is that the logic program orchestrates the constraints properly (e.g., for a single optimization problem, the atom $\texttt{use}_{\mathbf{c}}(c_{lsi})$ included as a fact would indicate that the constraint c_{lsi} should be used).

We can use the logic program to select the optimization criteria in a similar manner. We define two optimization criteria that we use in this paper as follows.

$$\begin{aligned}
o_{scr} :=&\quad \sum_{\substack{p \textit{ such that}\\ \{\texttt{assign}_{\mathbf{a}}(p,a),\\ \texttt{assign}_{\mathbf{f}}(p,f)\} \subseteq \Gamma^*(\Pi)}} 1 + scr(p) - trip(p,a,f)\\
o_{rtd} :=&\quad \sum_{\substack{p \textit{ such that}\\ \{\texttt{assign}_{\mathbf{a}}(p,a),\\ \texttt{assign}_{\mathbf{f}}(p,f)\} \subseteq \Gamma^*(\Pi)}} 1 + \frac{1}{1+rtd(p)} - trip(p,a,f)
\end{aligned}$$

Here, o_{scr} is an objective function resembling urgency based triage in that we want to increase the number of personnel evacuated and prioritize a higher individual triage score, while attempting to reduce total travel time. Likewise, o_{rtd} is similar, except that it prioritizes return-to-duty (hence $scr(p)$ is replaced with $\frac{1}{1+rtd(p)}$. Again, with a single optimization problem, $\texttt{use}_{\mathbf{o}}(o_{rtd})$, as seen as a fact in the last line of Table 1, would mean that we use the reverse triage optimization criteria. Note that again, with multiple optimization problems, we would use binary version of $\texttt{use}_{\mathbf{o}}$ (with the second argument being a natural number for the associated optimization problem). We note that the result of the deduction process gives us an optimization problem of the form "optimize o such that $\texttt{use}_{\mathbf{o}}(o) \in \Gamma^*(\Pi)$ subject to all c such that $\texttt{use}_{\mathbf{c}}(c) \in \Gamma^*(\Pi)$." We provide examples of how we use the logic to create integer programs in an online appendix [1]. However, the procedure is straightforward, as all constraints and optimization criteria are linear combinations over the integers zero and one.

From a user perspective, setting the facts with predicates $\texttt{use}_{\mathbf{o}}, \texttt{use}_{\mathbf{c}}$ will be entered in the GuardianTwin user interface in view called "Startegy Builder" shown in Figure 3 (left)). Based on the user's selections, facts formed with these two predicates are added for one or more optimization problems.

Reasoning Over Solutions to Optimization Problems. The PyReason implementation for logic programming allows us to maintain the logic program in memory and even add additional facts to it. As a result, we can follow the procedure of running the deductive process for logic program Π and for each

[1] https://tinyurl.com/appendixiclp2025

o such that there is some $\texttt{use}_{\mathbf{o}}(o,i) \in \Gamma^*(\Pi)$ we solve the associated optimization problem, where the result can be translated into a set of facts (e.g., $\Pi_{o,i}$). Then, we run the deduction process on that result that is equivalent to the following:

$$\Gamma^*(\Pi \cup \bigcup_{\substack{o \text{ such that there exists } i \text{ where} \\ \texttt{use}_{\mathbf{o}}(o,i) \in \Gamma^*(\Pi)}} \Pi_{o,i})$$

We note that the new facts added (e.g., each set $\Pi_{o,i}$) will cause additional rules in Π to fire to properly select among the solutions to the optimization problems. For example, we may want to select an optimization problem that is within a certain percent of optimal of urgency based triage while still honoring as many constraints as possible. Alternatively, the explainable trace of logic produced by the deductive process can also be shown to a user to allow them to make further decisions. We have started prototyping user interface designed to communicate the results of different optimization problems back to the user in GuardianTwin (Figure 3 (right))). It is important to note that end users of GuardianTwin—such as field medics or coordination officers will not interact with logic programs or optimization parameters directly. Instead, they engage with the system through high-level interface components like the Strategy Builder Figure 3 (left)), where abstract operational goals are translated automatically into logical facts. This will shield users from underlying complexity and ensure usability under high-stress conditions.

4 Experiments

Experimental Setup. Due to the sensitive nature of military operations, it is impossible to obtain real-world data for our experiments. To address this limitation, we evaluate our approach using an exercise-grade dataset created within the GuardianTwin framework. This dataset is engineered to capture realistic MEDEVAC operational scenarios by incorporating geographical constraints, casualty distributions, and asset limitations. The generated casualties, assets, and care facilities are designed to align closely with real-world parameters. The use of this exercise-grade dataset provides valuable insights into the potential performance of GuardianTwin in operational conditions. This dataset comprises three components: casualties, assets, and care facilities. Each casualty is represented as a tuple of the form $\langle name, AIS_1, \ldots, AIS_n, systolic\ blood\ pressure,\ respiratory\ rate, GCS, location \rangle$. Here AIS_i is the severity score of insult i with possible values in $\{0,1,2,3,4,5,6\}$. This vector representation of the constant allows us to easily implement many of the functions described in the previous section. Likewise, assets are represented by tuples of the form $\langle name, location, range, speed, remaining\ crew\ duty\ hours \rangle$ and care facilities are represented as tuples of the form $\langle name, location \rangle$. The locations of all casualties, assets, and care facilities are generated within a geographical area extending from the southern border of Arizona to the northern border of Utah, and from the western border of Arizona to the eastern border of Colorado. For each experimental run, we randomly select n casualties, k assets, and m care facilities from the provided dataset, with the goal of optimally assigning each casualty to an asset and care facility in order to maximize different criteria (as orchestrated by the logic program). To assess the advantages of our approach, we carried out multiple experiments to demonstrate abilities of our framework for MEDEVAC mission planning. We analyzed a sample of 250 data points. These data points were created using 10 samples from each of the following settings: $\{n = 25, m = 10, k = 1\}$, $\{n = 25, m = 10, k = 2\}$, $\ldots \{n = 25, m = 10, k = 25\}$. We utilized PyReason [1] to support logic programming inference and the mip package [8] for integer program optimization. Experiments were run on a MacBook equipped with 12-core CPU, 18-core GPU, and 18 GB unified memory. All optimization problems solved within an average of 6 milliseconds per scenario across varying numbers of assets and casualties, with none

exceeding 10 milliseconds. The logic program deduction steps completed in 3 to 5 seconds on all runs, demonstrating practical tractability for near real-time decision support. The code and data used in the experiments is available at `https://github.com/jaikrishna-patil/guardian-twin-opt`.

Exercise-Grade Data for Validation. The sensitive nature of military operations makes it impossible to collect real-world data. Our evaluation overcomes this limitation by using exercise-grade datasets which were specifically created to simulate real-world MEDEVAC situations and thus offer a strong operational testing environment for our framework. The datasets combine realistic elements like casualty profiles with geographic constraints and asset capabilities. Casualties possess attributes that include injury severity and vital signs along with geographic location and urgency levels which are assessed through established scoring systems like NISS, RTS, and LIFE. The dataset contains assets with operational details such as range capabilities and speed specifications along with base locations and remaining duty hours while care facilities are placed geographically to match real-world locations. The exercise-grade environment evaluates spatial factors such as distance measurements, terrain obstacles, and time-sensitive limitations to match real-world operation requirements. Expert knowledge and research studies about military and humanitarian MEDEVAC activities shaped the dataset design to achieve realistic scenarios with different casualty volumes and operational challenges. This data is not a contribution to this work; it was developed separately to support military exercises. This framework achieves practical applicability through simulation of real-world complex constraints while maintaining theoretical robustness and paving the way for live deployment upon obtaining required permissions.

Optimization Methods and Baselines. We use the logic program to orchestrate three different types of optimization problems which we investigate both individually and collectively by reasoning over different solutions. For comparison purposes, we also describe three basline models.

Urgency based Optimization. This optimization problem uses optimization criteria o_{scr}, constraint c_{lsi}, and constraint c^k_{scr} for values $k = \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$.

Reverse Triage Optimization. This optimization problem uses optimization criteria o_{rtd}, constraint c_{lsi}, and constraint c^k_{rtd} for values $k = \{50, 40, 30, 20, 10, 0\}$.

Situational Triage Optimization. This optimization problem uses optimization criteria o_{scr}, constraint c_{lsi}, constraint c^k_{air} for $k = 1$

Random Baseline Model (B1). In this baseline model, casualties are selected randomly and assigned to a randomly chosen available asset and care facility while adhering to constraint c_{lsi}.

Triage Priority Baseline Model (B2). In this model, casualties are selected one by one in descending order of *scr* and randomly assigned an available asset and care facility while adhering to constraint c_{lsi}.

Return to Duty Priority Baseline Model (B3). This model prioritizes casualties by ascending order of *rtd* and randomly assigned an available asset and care facility while adhering to constraint c_{lsi}.

These baselines were designed to mirror practical and interpretable heuristics inspired by real-world triage decision-making. While several reinforcement learning (RL) and machine learning (ML) approaches exist for triage and humanitarian logistics (see Section 2), such methods are typically trained to optimize a fixed criterion, often without constraint reasoning or user-in-the-loop customization. As our logic framework supports constraint-based orchestration and multi-objective selection, direct empirical comparison to black-box learning methods is not straightforward. We therefore use structured internal baselines that align with clinical heuristics.

Single Optimization Problem. Figure 4 (left) demonstrates that the assignments provided by urgency based and reverse optimization problems resulted in more casualties being evacuated than any of the

baselines or situational optimization across nearly all asset levels (in that figure, we use 25 casualties and 10 facilities). On average, urgency based and reverse optimization assignments evacuated more than twice the casualties evacuated by B1 and B2. This improvement goes to 61.32% increase when compared with B3. We also observe that number of casualties evacuated by assignments provided by situational optimization problem underperformed relative to our other methods. This makes sense because latter optimization problem has additional air time constraint to mitigate asset risk. In Figure 4 (right) we examine the sum of the triage score for evacuated casualties. Here, we also see the best performance is obtained with urgency based and reverse optimization - though they diverge more than in the earlier experiment.

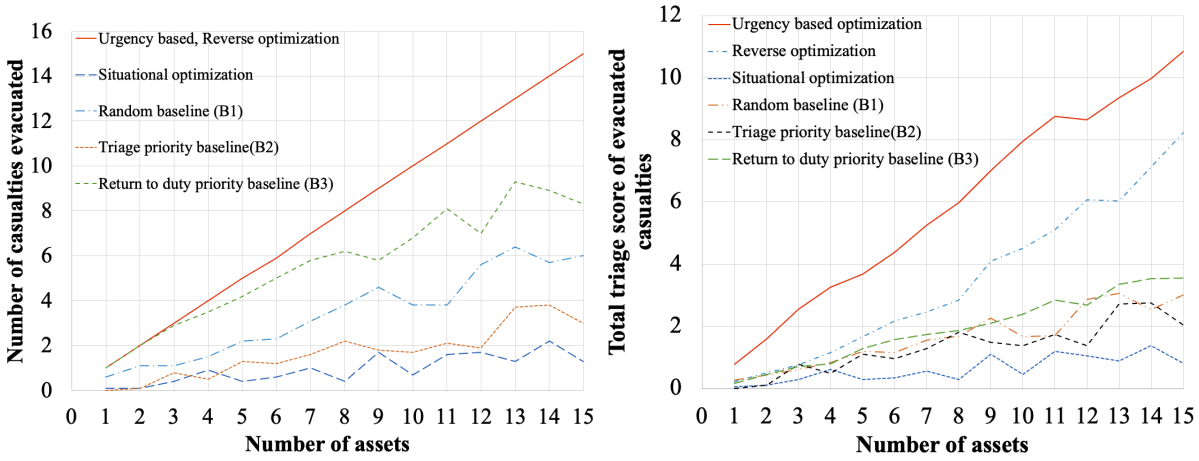


Figure 4: Number of casualties evacuated successfully (left) and total traige score of all evacuated casualties (right) by single optimization problems and baseline models for different number of assets.

Multiple Optimization Problems. A key aspect of our framework is the ability to examine the results of multiple optimization problems simultaneously. We investigate this from two different respects. First, we look at relative performance of an optimization problem with respect to other optimization criteria - which can provide insight into the development of a logic program to select among different optimization criteria. The second deals with constraint relaxation or removal. In this case, particularly with constraint that use a threshold (e.g. c^k_{scr}) we may not have a feasible solution to the optimization problem, so instead we relax constraints (via rules in logic program Π) and re-stage new optimization problems. In Figure 5 we examine the performance of a given optimization problem staged by Π with respect to other optimization criteria. In the left-hand panel, we see, for example, that the Reverse Triage Optimization problem comes within 66% of the value of o_{scr} for several asset ranges (e.g., for 7 assets) while providing a 4-fold increase over the Urgency based Triage Optimization problem for o_{rtd}. This type of understanding, can be, in-turn leveraged by the logic program after facts created from the optimization problem are placed back in the logic program. A natural way to consider this is to have logical rules that relax or remove constraints. In Figure 6 we consider this where we relax k values for constraints c^k_{scr} (left) and c^k_{rtd} (right) using rules in Π that reason about the results of multiple optimization problems. We observe in both cases that for smaller number of available assets i.e. $k = \{1,2,3,4,6\}$, constraint needs to be fully relaxed in order to get a feasible solution. However, as number of available assets increase, we are able to get feasible solutions while still adhering to a subset of the constraints.

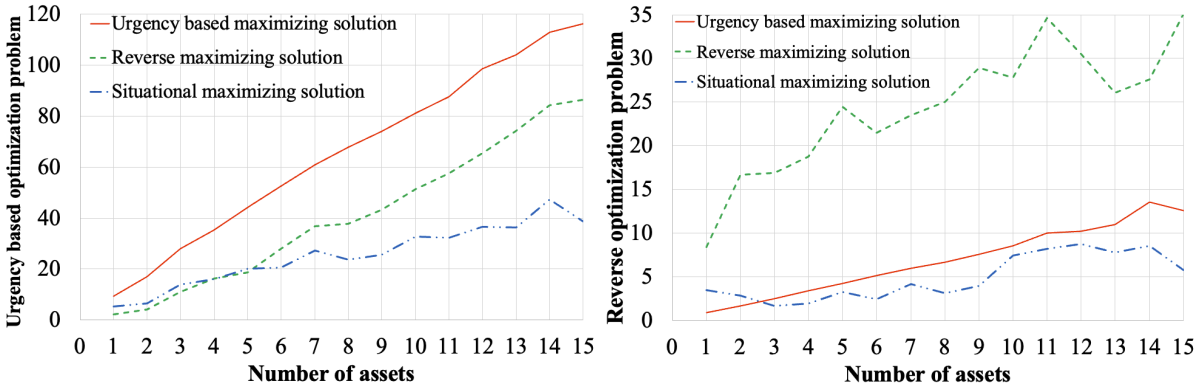


Figure 5: Performance of urgency based, reverse and situational optimization solution on urgency based optimization criteria (left) and reverse triage optimization criteria (right).

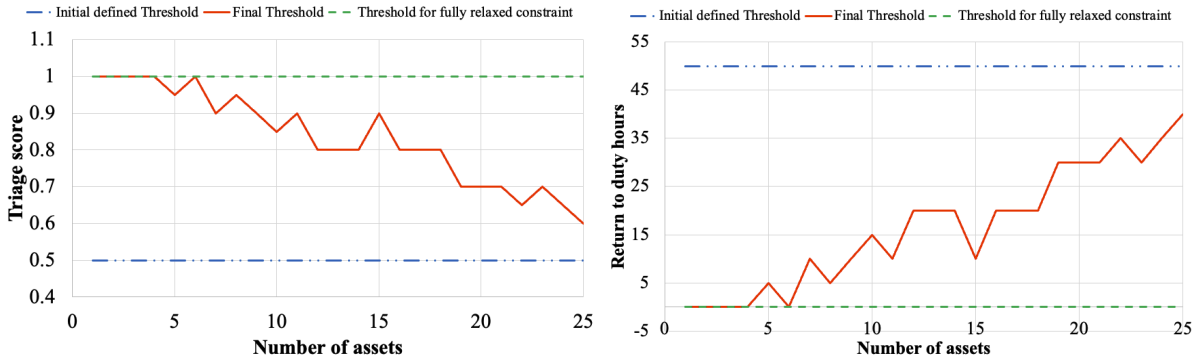


Figure 6: Left: Relaxation of constraint with triage score. Right: Relaxation of constraint with return to duty hours.

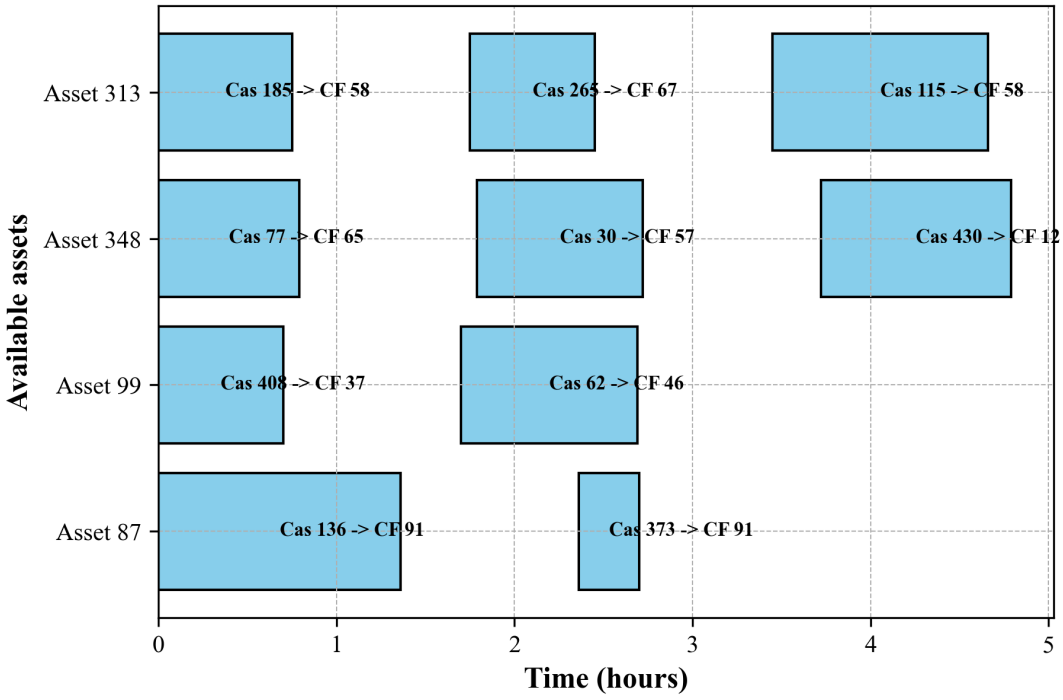


Figure 7: Sample Iterative schedule.

Orchestrating sequential optimization problems. In real MEDEVAC scenarios, assets are assigned to casualties and then re-assigned upon completion of their mission. As our framework allows for the iterative orchestration of optimization problems, we can leverage it to orchestrate sequences of optimization problems over time. In Figure 7 we show a Gantt chart resulting from this iterative process. As part of future work, we look to leverage the temporal aspects of the underlying logic [1] to allow for additional modeling (e.g., changes to the crew over time, enemy situation, etc.) as well as explore the use of planning to examine optimal solutions over multiple time steps.

5 Conclusion

In this paper, we showed how we could use logic programming to orchestrate one or more optimization problems for medical triage resource allocation and then, in turn, reason about the results. Such reasoning can be used to select the results of a particular optimization problem or to sequence multiple optimization problems over time. We show how the symbolic representations of the logic program provide a natural analogue for a user interface - allowing the user to customize the logic program and stage optimization problems in various ways. As we continue the development of GuardianTwin and move toward deployment, we shall refine how the user can benefit from automated decisions to reason over the results of the optimization problems, explore trade-offs in medical triage, as well as incorporate additional knowledge such as data about the enemy and other characteristics of medical treatment.

Path to Deployment. Due to the sensitive nature of military MEDEVAC operations, obtaining real-world data or immediate access to live deployment scenarios is not feasible. Instead, GuardianTwin is evaluated using exercise-grade datasets generated within the framework itself, which are designed to simulate realistic casualty profiles, resource constraints, and geographic challenges. This approach enables rigorous, reproducible testing under operationally relevant conditions while adhering to security protocols. The deployment pathway follows a structured, three-phase strategy: initial validation using simulated scenarios and expert feedback; beta testing in controlled environments with field medics and coordinators; and eventual deployment in live operations, subject to required security clearances and institutional approval. This phased progression ensures both technical reliability and practical readiness for real-world use.

References

[1] Dyuman Aditya, Kaustuv Mukherji, Srikar Balasubramanian, Abhiraj Chaudhary & Paulo Shakarian (2023): *PyReason: Software for Open World Temporal Logic*, doi:10.48550/arXiv.2302.13482. arXiv:2302.13482.

[2] Michael D April, Rachel E Bridwell, Jessica Jones, Joshua Oliver, Brit Long & Steven G Schauer (2022): *Descriptive Analysis of Casualties Rapidly Returned to the Fight after Injury: Reverse Triage Implications for Large Scale Combat Operations. Medical Journal, US Army Medical Center of Excellence (MEDCoE).*

[3] M.C.M. Bricknell & A.G. Johnson (2011): *Forward Medical Evacuation*. *Journal of the Royal Army Medical Corps* 157, pp. S444–S448, doi:10.1136/jramc-157-4s-07.

[4] Michael Coslovsky, Jukka Takala, Aristomenis K Exadaktylos, Luca Martinolli & Tobias M Merz (2015): *A clinical prediction model to identify patients at high risk of death in the emergency department*. *Intensive care medicine* 41, pp. 1029–1036, doi:10.1007/s00134-015-3737-x.

[5] Marta Fernandes, Susana M Vieira, Francisca Leite, Carlos Palos, Stan Finkelstein & João MC Sousa (2020): *Clinical decision support systems for triage in the emergency department using intelligent systems: a review*. *Artificial Intelligence in Medicine* 102, p. 101762, doi:10.1016/j.artmed.2019.101762.

[6] Andrew G Gelbard, Phillip R Jenkins & Matthew J Robbins (2024): *Enhancing military medical evacuation dispatching with armed escort management: comparing model-based reinforcement learning approaches*. The Journal of Defense Modeling and Simulation, p. 15485129241229762, doi:10.1177/15485129241229762.

[7] Vahid Ghanbari, Ali Ardalan, Armin Zareiyan, Amir Nejati, Dan Hanfling, Alireza Bagheri & Leili Rostamnia (2021): *Fair prioritization of casualties in disaster triage: a qualitative study*. BMC emergency medicine 21, pp. 1–9, doi:10.1186/s12873-021-00515-2.

[8] Túlio A.M. Toffolo Haroldo G. Santos (2020): *Mixed Integer Linear Programming with Python*. Available at `https://readthedocs.org/projects/python-mip/downloads/pdf/latest/`.

[9] Champion Hr (1989): *A revision of the Trauma Score*. J Trauma 29, pp. 623–629, doi:10.1097/00005373-198905000-00017.

[10] Phillip R Jenkins, Brian J Lunday & Matthew J Robbins (2020): *Robust, multi-objective optimization for the military medical evacuation location-allocation problem*. Omega 97, p. 102088, doi:10.1016/j.omega.2019.07.004.

[11] Sarah Johnson, Ryann S Lauby, Michael D April, Andrew D Fisher, Joseph K Maddry, Luk Hofmann, Kevin Gillespie, Jose Salinas & Steven G Schauer (2022): *Development of Data-Driven Triage Systems for Identifying Mortally Wounded Casualties–Implications for Future Large-Scale Combat Operations*. Medical Journal, US Army Medical Center of Excellence (MEDCoE).

[12] Michael Kifer & V.S. Subrahmanian (1992): *Theory of generalized annotated logic programming and its applications**A preliminary report on this research has appeared in [34]*. The Journal of Logic Programming 12(4), pp. 335–367, doi:10.1016/0743-1066(92)90007-P. Available at `https://www.sciencedirect.com/science/article/pii/074310669290007P`.

[13] Sathvika Kotha, Hrishikesh Viswanath, Kshitij Tiwari & Aniket Bera (2023): *ARTEMIS: AI-driven Robotic Triage Labeling and Emergency Medical Information System*. arXiv preprint arXiv:2309.08865, doi:10.48550/arXiv.2309.08865.

[14] Seunghoon Lee & Young Hoon Lee (2020): *Improving Emergency Department Efficiency by Patient Scheduling Using Deep Reinforcement Learning*. Healthcare 8(2), doi:10.3390/healthcare8020077. Available at `https://www.mdpi.com/2227-9032/8/2/77`.

[15] Turner Osler, Susan P Baker & William Long (1997): *A modification of the injury severity score that both improves accuracy and simplifies scoring*. Journal of Trauma and Acute Care Surgery 43(6), pp. 922–926, doi:10.1097/00005373-199712000-00009.

[16] Yoshihiko Raita, Tadahiro Goto, Mohammad Kamal Faridi, David FM Brown, Carlos A Camargo & Kohei Hasegawa (2019): *Emergency department triage prediction of clinical outcomes using machine learning models*. Critical care 23, pp. 1–13, doi:10.1186/s13054-019-2351-7.

[17] Channel A. Rodriguez, Phillip R. Jenkins & Matthew J. Robbins (2023): *Solving the joint military medical evacuation problem via a random forest approximate dynamic programming approach*. Expert Systems with Applications 221, p. 119751, doi:10.1016/j.eswa.2023.119751. Available at `https://www.sciencedirect.com/science/article/pii/S095741742300252X`.

[18] Paulo Shakarian & Gerardo I. Simari (2022): *Extensions to Generalized Annotated Logic and an Equivalent Neural Architecture* . In: 2022 Fourth International Conference on Transdisciplinary AI (TransAI), IEEE Computer Society, Los Alamitos, CA, USA, pp. 63–70, doi:10.1109/TransAI54797.2022.00017.

[19] Manuel Sigle, Leon Berliner, Erich Richter, Mart van Iersel, Eleonora Gorgati, Ives Hubloue, Maximilian Bamberg, Christian Grasshoff, Peter Rosenberger, Robert Wunderlich et al. (2023): *Development of an anticipatory triage-ranking algorithm using dynamic simulation of the expected time course of patients with trauma: modeling and simulation study*. Journal of Medical Internet Research 25(1), p. e44042, doi:10.2196/44042.

[20] Scott C. Sundstrom, Christopher G. Blood & Serge A. Matheny (1996): *The optimal placement of casualty evacuation assets: a linear programming model*. In: *Proceedings of the 28th Conference on Winter Simulation*, WSC '96, IEEE Computer Society, USA, p. 907–911, doi:10.1145/256562.256847.

[21] Graham Teasdale & Bryan Jennett (1974): *Assessment of coma and impaired consciousness: a practical scale*. *The Lancet* 304(7872), pp. 81–84, doi:10.1016/s0140-6736(74)91639-0.

[22] Zhou Yang, Long Nguyen, Jiazhen Zhu, Zhenhe Pan, Jia Li & Fang Jin (2020): *Coordinating Disaster Emergency Response with Heuristic Reinforcement Learning*. In: *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 565–572, doi:10.1109/ASONAM49781.2020.9381416.

[23] Lina Yu, Canrong Zhang, Jingyan Jiang, Huasheng Yang & Huayan Shang (2021): *Reinforcement learning approach for resource allocation in humanitarian logistics*. *Expert Systems with Applications* 173, p. 114663, doi:10.1016/j.eswa.2021.114663.

The ASP-based Nurse Scheduling System at the University of Yamanashi Hospital

Hidetomo Nabeshima
University of Yamanashi, Japan

Mutsunori Banbara
Nagoya University, Japan

Torsten Schaub
University of Potsdam, Germany

Takehide Soh
Nagoya University, Japan

We present the design principles of a nurse scheduling system built using Answer Set Programming (ASP) and successfully deployed at the University of Yamanashi Hospital. Nurse scheduling is a complex optimization problem requiring the reconciliation of individual nurse preferences with hospital staffing needs across various wards. This involves balancing hard and soft constraints and the flexibility of interactive adjustments. While extensively studied in academia, real-world nurse scheduling presents unique challenges that go beyond typical benchmark problems and competitions. This paper details the practical application of ASP to address these challenges at the University of Yamanashi Hospital, focusing on the insights gained and the advancements in ASP technology necessary to effectively manage the complexities of real-world deployment.

1 Introduction

The University of Yamanashi Hospital[1] serves as a core medical institution providing advanced medical care in the region and is one of the largest hospitals in the Yamanashi prefecture. It has approximately 620 beds, 400 physicians, and 760 nurses. The hospital consists of 19 wards, each with its own work schedule for the nurses assigned to it.

In our hospital, nurse schedules are manually created by head nurses, requiring significant effort and time. To automate the scheduling process, we formulated the Nurse Scheduling Problem (NSP) specific to our hospital. We first presented a typical NSP formulation for Japanese hospitals [6] to the head nurses. Based on their feedback, we refined the model by adding missing entities or constraints and removing redundant ones, thereby developing a base model. Subsequently, we focused on the obstetrics and gynecology wards, which have particularly strict requirements. With our NSP model, we automatically generated schedules for this ward. These schedules were then evaluated by the head nurse of the ward, who provided feedback by identifying unacceptable or unnatural elements. Based on this feedback, we refined the formulation of the NSP. We call this process the *iterative modeling refinement cycle*, which involves generating schedules, evaluating the results, and refining the formulation iteratively. This cycle was repeated over approximately one year, during which schedules were generated and refined every four weeks. The NSP formulation presented in this section represents the current state of the model, which has been tailored to meet the specific needs of our hospital.

Head nurses possess both explicit and implicit knowledge regarding nurse scheduling. While much of the explicit knowledge can be identified during the initial formulation of the NSP, implicit knowledge often emerges during the evaluation of automatically generated schedules. Therefore, the iterative modeling refinement cycle is essential for extracting implicit knowledge. However, fully capturing

[1] `https://www.hosp.yamanashi.ac.jp`

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 334–348, doi:10.4204/EPTCS.439.23

© Nabeshima et al.
This work is licensed under the
Creative Commons Attribution License.

Table 1: Nurse Shifts Targeted for Automatic Scheduling in the Hospital

Sign	Description	Work hours
EM	Early morning shift	06:00–14:45
D	Day shift	08:00–16:45
LD	Long day shift	08:00–20:15
LM	Late morning shift	11:30–20:15
E	Evening shift	16:00–00:45
SE	Short evening shift	19:45–00:00
N	Night shift	00:00–08:45
SN	Short night shift	00:00–08:45
WO	Weekly day off	—
PH	Public holiday leave	—

implicit knowledge remains a significant challenge. To address this, we acknowledge that our NSP formulation is inherently incomplete and propose a system to support modifications to the generated schedules, ensuring they better align with the head nurse's requirements. Details of this support system are provided in the next section.

The basic entities in our NSP model include *nurses*, *nurse groups*, *shifts*, *shift groups*, *dates*, *past shifts*, and *requested shifts* for the current and following months.[2] Nurses are characterized by their ID and skill level. Each nurse is assigned to exactly one ward. The number of nurses varies by ward, ranging from 15 to 40.[3] However, this number may fluctuate during the scheduling period due to factors such as new hires, retirements, or staff transfers. Nurse groups are sets of nurses classified based on their skills and experience.[4] Examples of such groups include senior, intermediate, and novice groups. Nurses may belong to multiple groups. For instance, head nurses and deputy head nurses may be part of both the senior and leadership groups.

Table 1 lists the shifts subject to automatic assignment, categorized into *work shifts* and *rest shifts*. Work shifts include scheduled hours such as day, evening, and night shifts, while rest shifts represent non-working days, including weekly rest and designated off days. In this study, rest days are also treated as shifts. In practice, the working hours of shifts vary across wards in the hospital, resulting in a larger number of distinct shifts. However, in this study, these variations are abstracted and consolidated into eight standardized work shifts for simplicity. The working hours shown in Table 1 represent typical time ranges. Standard NSP models typically include only day, evening, and night shifts. The inclusion of a larger variety of work shifts is one of the unique features of our NSP model. Additionally, there are two specific categories of shifts: *duty shifts* and *leave shifts*. Duty shifts represent special assignments such as business trips and training, while leave shifts cover various types of leave, including annual leave, maternity leave, and childcare leave. These shifts are assigned based on nurse requests and are not subject to automatic assignment. Therefore, they are not included in the table. Although N and SN have the same working hours, they are treated as separate shifts because they are typically assigned to different groups of nurses. This distinction is also reflected in shift patterns; for example, the SE shift is followed by SN, while the E shift is followed by N.

[2]Since the scheduling period is divided into four-week intervals, the terms "current" and "following months" refer to these four-week units.

[3]We focus on full-time nurses because part-time nurses in our hospital follow fixed work patterns.

[4]Nurse groups are one of the commonly used entities in NSP and correspond to "skills" in the INRC (The International Nurse Rostering Competition, `https://mobiz.vives.be/inrc2`) [4].

A shift group is a set of shifts. For example, the shift group {E, SE} is used to specify the required number of nurses for evening shifts. Constraints related to the number of nurses or the number of shift assignments are often defined on shift groups. This concept of shift groups is also a unique feature of our NSP model.

Dates include past shift dates, the current month, and the first week of the next month to ensure consistency across scheduling boundaries. Past shifts refer to schedules from periods prior to the current month. These schedules are necessary to ensure consistency between assignments at the end of the previous month and the start of the current month, and they are essential for maintaining equitable shift allocations. Requested shifts are classified as desired or undesired, and represented as a triplet of nurse, date, and shift. Shifts cover work, weekly rest, leave, and duty shifts. Including early next-month shifts helps preserve scheduling continuity. Notably, in our hospital, requested shifts are treated primarily as hard constraints.

The following is the list of constraints in our NSP model. Details of some constraints are presented later through their ASP encodings. The full ASP encoding for all constraints is available in the repository at `https://github.com/nabesima/yamanashi-nsp`. This repository also includes anonymized real instances, artificially generated instances of various sizes, and scripts for solving the problem.

Workdays (H_1): Nurses must be scheduled to work a specific number of days per scheduling period, typically 20 days. This number may decrease due to external factors such as business trips, public holidays, or paid leave.

Weekly Rest Days (H_2): Nurses must be assigned a specified number of weekly rest days, usually 8 days per period.

Requested shifts (H_3): Nurses must be assigned to their desired shifts while avoiding undesired shifts.

Average Working Hours (H_4): The assignments for LD and SE shifts must be equalized to ensure an average daily working time of 7 hours and 45 minutes.

Consecutive Workdays (H_5, S_1): Consecutive workdays beyond the specified limit, typically 5 or 6 days, are prohibited.

Daily Staffing (H_6, S_2): The required number of nurses must be ensured for each nurse group, each shift group, and each day. Alternatively, nurses must be assigned such that the total skill levels exceed the specified threshold.

Shift Frequency (H_7, S_3): Nurses are assigned shifts within the predefined range for each shift group to ensure balanced workloads. For example, nurses in the night shift group may be assigned 4–8 night shifts per period.

Shift Patterns (H_8, S_4): Work shift patterns must fall within the specified range, such as avoiding overly irregular schedules. For example, the shift pattern LD-D (a long day shift followed by a day shift) may be allowed at most once per period.

Inter-Shift Rules (H_9, S_5): Consecutive shifts must comply with the specified constraints. For instance, an evening shift must be followed by a night shift. It is recommended that the next shift begins at least 24 hours after the start of the previous shift.

Nurse Pairing (H_{10}, S_6): Recommended nurse pairs (e.g., mentor-novice pairs) should ideally be assigned the same shifts whenever possible, while prohibited pairs must not work the same shifts.

Isolated Workdays (S_7): Isolated single workdays surrounded by rest days should be minimized to reduce unnecessary disruptions.

Leave-Adjacent Rest Days (S_8): Weekly rest days should be scheduled before or after requested leave days to extend rest periods.

Equal Workload Distribution (S_9): Shift workloads should be distributed evenly among nurses to ensure fairness. For example, this includes the number of rest days on weekends or public holidays.

These constraints are classified into *hard constraints* (H_i), which must always be satisfied, and *soft constraints* (S_i), which should be satisfied as much as possible. Hard constraints such as H_1, H_2, and H_5–H_8 enforce strict lower and/or upper bounds, while soft constraints such as S_1–S_4 and S_6 define desirable limits. Violating a soft constraint incurs a penalty proportional to the square of the deviation from the threshold to discourage large violations. Other soft constraint violations incur a fixed penalty per occurrence. A *feasible solution* is a shift assignment to each nurse and each day that satisfies all hard constraints. Soft constraints are prioritized, with their importance varying by ward. The objective is to find a feasible solution that minimizes penalty costs in lexicographical order based on these priorities. Additionally, some wards have specific constraints. For example, in the ICU, six consecutive workdays are allowed, but new hires are limited to five, and their night shift assignments must be equal.

2 ASP Encoding of Constraints

The ASP encoding of our NSP model consists of approximately 100 rules. Due to space limitations, only a subset of the fundamental constraints is presented here. The complete encoding is available in our repository. Listing 1 shows the ASP encoding for constraints H_1–H_7 and S_2–S_3, with certain constraints omitted for brevity. Some predicates appearing in the figure are either directly defined in the instance or generated during preprocessing.

Workdays (H_1, lines 2–7): Fact `staff(N)` specifies that `N` is a nurse, and `work_shift(S)` indicates that `S` is a work shift. Similarly, `workable_date(N,D)` indicates that `N` is available to work on day `D`, meaning `N` is not assigned to a duty shift on that day. The predicates `assigned(N,D)` and `assigned(N,D,S)` define the schedule, with the former indicating that `N` works on `D` and the latter specifying the assigned shift `S`. Line 2 ensures assignments only on `workable_date(N,D)`, while line 3 enforces exactly one shift per workday. Lines 4–7 constrain the number of workdays within the bounds `LB` and `UB` specified by `work_days_bounds(N,LB,UB)`. Constraint violations are tracked using `violation(T,C,LIM,VAL)`, where `T` is the violation type (`hard` or `soft`), `C` is the reason, and `LIM` and `VAL` denote the target and actual values. This predicate is discussed below.

Weekly Rest Days (H_2, lines 10–13): These rules ensure that the number of weekly rest days assigned to each nurse `N` falls within the range specified by `weekly_rest_bounds(N,LB,UB)`. The fact `weekly_rest_available_date(N,D)` indicates that `N` can be assigned a weekly rest day on `D`, provided `D` is neither a public holiday nor a day with a paid leave request.

Requested Shifts (H_3, lines 16–19): The fact `pos_request(N,D,S)` represents that nurse `N` has a desired shift `S` on day `D`, and multiple desired shifts can be specified for the same day. `pos_request(N,D)` denotes that `N` has at least one desired shift on `D`. The predicate `ext_assigned(N,D,S)` extends `assigned(N,D,S)` to cover all shift types, not just work shifts. Details are explained shortly. Lines 16–17 ensure at least one desired shift is assigned. Similarly, `neg_request(N,D,S)` represents shifts that nurse `N` prefers to avoid, and lines 18–19 ensure they are not assigned. The predicate `violation(T,C)` represents a constraint violation with a fixed penalty, while its four-argument version includes additional parameters for violation severity.

The predicate `ext_assigned(N,D,S)` is defined on lines 22–26. If `assigned(N,D,S)` holds, then `ext_assigned(N,D,S)` also holds (line 22). If nurse `N` has no assigned work shift on day `D`, one of the following shifts is assigned: (1) the weekly rest shift, (2) the public holiday shift, or (3) a duty or leave shift requested by the nurse. The facts `weekly_rest_avail_date(N,D)` and `pub_holiday_avail_date(N,D)` indicate that nurse `N` has no duty or leave requests on day `D` and that the day is either a non-holiday or a holiday, respectively. The former corresponds to (1), where the

weekly rest shift (WR) is assigned, while the latter corresponds to (2), assigning the public holiday shift (PH) (lines 22–23). The fact `manual_request(N,D,S)` represents a nurse's request for a duty or leave shift on day `D`, corresponding to (3). In this case, exactly one of the requested duty or leave shifts is assigned (lines 25–26).

Average Working Hours (H_4, lines 29–30): This rule enforces that subtracting the number of SE shifts from the number of LD shifts assigned to each nurse `N` results in zero, ensuring that the two shift types are assigned equally.

Consecutive Workdays (H_5, S_1, lines 33–38): The fact `consecutive_work_ub(T, NG,UB)` specifies the upper bound `UB` on consecutive workdays for nurse group `NG`, where `T` denotes the constraint type (`hard` or `soft`). In our NSP model, no lower bound is imposed. The fact `staff_group(NG,N)` indicates that nurse `N` belongs to nurse group `NG`, and `base_date(D)` includes days in the current month and one week before and after, accounting for consecutive work periods spanning adjacent months. The predicate `work_day(N,D)` indicates that `N` has either a work shift or a duty shift on day `D`. Lines 33–35 define the predicate `full_work_period(N,BD,ED)` when a nurse works exactly up to the upper bound of consecutive workdays, with `BD` and `ED` as the start and end dates. Lines 36–38 identify a violation if the upper bound is exceeded by one day. If exceeded by n days, n violations are recorded.

Daily Staffing (H_6, S_2, lines 41–50): The fact `shift_group(SG,S)` specifies that shift `S` belongs to shift group `SG`, and `staff_lb(T,NG,SG,D,LB)` gives the lower bound `LB` on the number of nurses from group `NG` required for shift group `SG` on day `D`. Lines 41–42 define violations when this bound is unmet. The same applies to upper bounds. The fact `point_lb(T,NG,SG,D,LB)` gives the lower bound on the total skill levels of nurses, and `point(N,P)` specifies that nurse `N` has skill level `P`. Lines 45–47 define violations when total skill levels fall below `LB`. Upper bounds are handled similarly. Some wards impose constraints solely on nurse headcount, while others enforce both headcount and skill-level constraints. In the latter case, satisfying either constraint is sufficient. This is implemented by defining a rule that treats a violation as occurring only when both constraints are violated simultaneously. The detailed encoding is omitted due to space limitations.

Shift Frequency (H_7, S_3, lines 53–56): The fact `shift_lb(T,N,SG,LB)` specifies that the total number of shifts assigned to nurse `N` from shift group `SG` must be at least `LB`, where `T` denotes the constraint type. Lines 53–54 ensure that this total meets or exceeds `LB`. Upper bounds are enforced similarly.

Listing 2 defines the treatment of constraint violations and the objective function. The predicate `soften_hard` is used to relax hard constraints. Hospital wards often face nursing staff shortages due to scheduled absences (e.g., training, business trips) and unexpected long-term absences (e.g., illness, injury). Additionally, idealistic staffing requirements set by head nurses can further restrict shift flexibility. These factors frequently lead to constraint violations, making it difficult to fully adhere to hard constraints. To address this, a mechanism for relaxing hard constraints is essential. By default, `soften_hard` is set to false. If the NSP instance is unsatisfiable, enabling it allows schedule generation despite hard constraint violations. The first line defines its choice rule, with its truth value provided as an assumption in incremental ASP solving [7]. This mechanism enhances response time by allowing the solver to resume search immediately instead of redoing both grounding and search when toggling `soften_hard`. Lines 2 and 3 enforce that hard constraint violations are prohibited unless `soften_hard` is true.

Lines 5–7 encode penalty calculations for constraint violations. The predicate `penalty(T,C,W,P)` represents a penalty, where `T` denotes the constraint type, `C` the penalty reason, `W` the weight, and `P` the priority. The fact `priority(T,C,P)` assigns priority `P` to reason `C` with the constraint type `T`. If a violation has a severity level, its squared value is used as the penalty weight (lines 5–6); otherwise, a constant weight is applied (line 7). The objective function, defined on line 9, minimizes the sum of penalty weights in lexicographical order based on priority.

```
% H1. Workdays
{ assigned(N,D) : workable_date(N,D) } :- staff(N).
1 { assigned(N,D,S) : work_shift(S) } 1 :- assigned(N,D).
violation(hard,work_days_lb(N),LB,X) :-
    staff(N), work_days_bounds(N,LB,UB), X = { assigned(N,D) }, X < LB.
violation(hard,work_days_ub(N),UB,X) :-
    staff(N), work_days_bounds(N,LB,UB), X = { assigned(N,D) }, UB < X.

% H2. Weekly Rest Days
violation(hard,weekly_rest_lb(N),LB,X) :- weekly_rest_bounds(N,LB,UB),
    X = { not assigned(N,D) : weekly_rest_available_date(N,D) }, X < LB.
violation(hard,weekly_rest_ub(N),UB,X) :- weekly_rest_bounds(N,LB,UB),
    X = { not assigned(N,D) : weekly_rest_available_date(N,D) }, UB < X.

% H3. Requested Shifts
violation(hard,pos_request(N,D)) :- pos_request(N,D), date(D),
    not 1 { ext_assigned(N,D,S) : pos_request(N,D,S) } 1.
violation(hard,neg_request(N,D)) :- neg_request(N,D), date(D),
    1 { ext_assigned(N,D,S) : neg_request(N,D,S) }.

% Extends assigned/3 to ext_assigned/3
ext_assigned(N,D,S) :- assigned(N,D,S).
ext_assigned(N,D,"WR") :- not assigned(N,D), weekly_rest_avail_date(N,D).
ext_assigned(N,D,"PH") :- not assigned(N,D), pub_holiday_avail_date(N,D).
1 { ext_assigned(N,D,S) : manual_request(N,D,S) } 1 :-
    not assigned(N,D), manual_request(N,D).

% H4. Average Working Hours
violation(hard,eq_shifts(N)) :- staff(N),
    not 0 #sum{ 1,D : assigned(N,D,"LD") ; -1,D : assigned(N,D,"SE") } 0.

% H5. Consecutive Workdays
full_work_period(N,BD,ED) :- consecutive_work_ub(_,NG,UB), staff(N),
    staff_group(NG,N), base_date(BD), ED=BD+UB-1, base_date(ED), -1 <= ED,
    work_day(N,D) : D = BD..ED.
violation(T,consecutive_work_days(N, BD),UB,UB+1) :-
    consecutive_work_ub(T,NG,UB), staff_group(NG,N),
    full_work_period(N,BD,ED), work_day(N,ED+1).

% H6, S2. Daily Staffing
violation(T,staff_lb(NG,SG,D),LB,X) :- staff_lb(T,NG,SG,D,LB),
    X = { assigned(N,D,S) : staff_group(NG,N), shift_group(SG,S) }, X < LB.
violation(T,staff_ub(NG,SG,D),UB,X) :- staff_ub(T,NG,SG,D,UB),
    X = { assigned(N,D,S) : staff_group(NG,N), shift_group(SG,S) }, UB < X.
violation(T,point_lb(NG,SG,D),LB,X) :- point_lb(T,NG,SG,D,LB),
    X = #sum{ P,N : point(N,P), assigned(N,D,S),
                    staff_group(NG,N), shift_group(SG,S) }, X < LB.
violation(T,point_ub(NG,SG,D),UB,X) :- point_ub(T,NG,SG,D,UB),
    X = #sum{ P,N : point(N,P), assigned(N,D,S),
                    staff_group(NG,N), shift_group(SG,S) }, UB < X.

% H7, S3. Shift Frequency
violation(T,shift_lb(N,SG),LB,X) :- shift_lb(T,N,SG,LB),
    X = { assigned(N,D,S) : shift_group(SG,S) }, X < LB.
violation(T,shift_ub(N,SG),UB,X) :- shift_ub(T,N,SG,UB),
    X = { assigned(N,D,S) : shift_group(SG,S) }, UB < X.
```

Listing 1: ASP Encoding of Basic Constraints

```
{ soften_hard }.
:- violation(hard,_,_,_), not soften_hard.
:- violation(hard,_), not soften_hard.

penalty(T,C,W,P) :- violation(T,C,LIM,VAL),
  W = (LIM-VAL)*(LIM-VAL), priority(T,C,P).
penalty(T,C,1,P) :- violation(T,C), priority(T,C,P).

#minimize { W@P,T,C : penalty(T,C,W,P) }.
```

Listing 2: ASP Encoding for Constraint Violations and Objective Function.

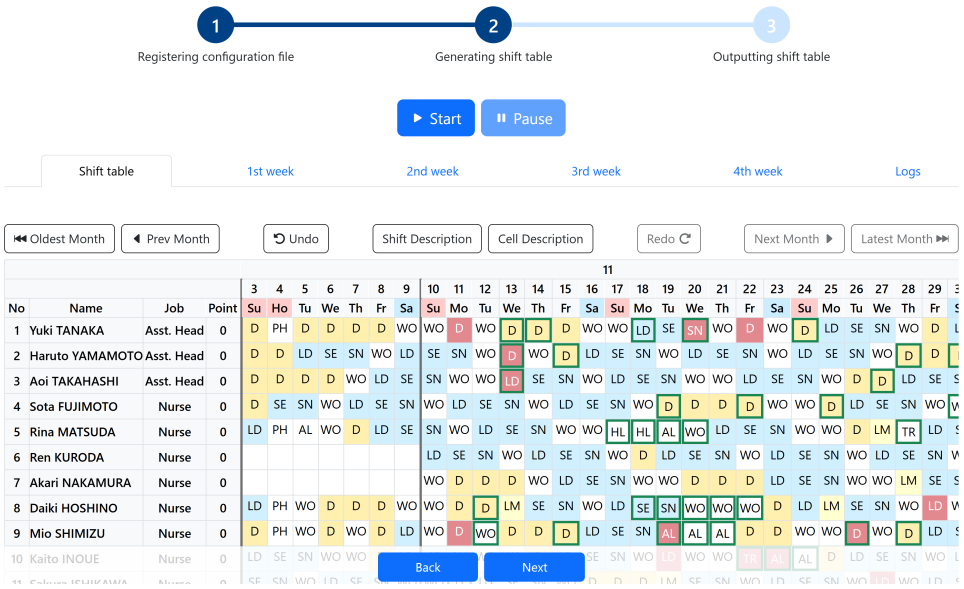


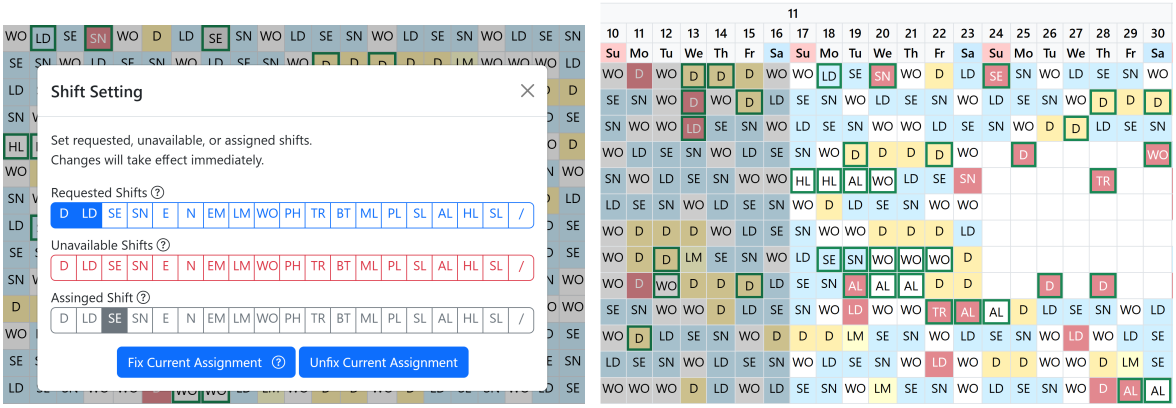
Figure 1: Schedule generation interface in *aspital*.

3 Nurse Scheduling System

Based on the NSP model presented in the previous section, we developed *aspital*, a system for nurse schedule generation. *aspital* is provided as a web service, allowing head nurses to generate and modify schedules through a browser interface.[5] Fig. 1 shows an example of its schedule generation interface. In the backend, it utilizes the ASP solver *clingo* to solve the NSP and updates the schedule immediately whenever a solution changes. A key feature of *aspital* is that it not only generates schedules automatically but also provides tools to assist head nurses in manual adjustments. Further details are given below. The system is currently being trialed in six wards of our hospital.

The automatically generated schedule does not always fully satisfy all head nurse requirements due to the difficulty of extracting their implicit knowledge, often resulting in an incomplete schedule. Additionally, reviewing the schedule is a time-consuming and labor-intensive task, yet its burden is often underestimated, making the execution of the iterative modeling refinement cycle challenging in real-world operations. Given that the generated schedule should already meet most of the head nurse's requirements, modifying only the unacceptable parts is a more practical approach. Furthermore, even after a schedule is

[5]Only the command-line version is available in our public repository.



(a) Manual shift modifications

(b) Fixing and clearing shift assignments

Figure 2: Shift editing interface in *aspital*

finalized, rescheduling is often required due to unforeseen circumstances, such as sudden absences caused by illness. Therefore, providing a mechanism to support schedule modifications is essential in practice.

To address these challenges, *aspital* provides various features for schedule modifications, enabling head nurses to make targeted adjustments efficiently. If a schedule generated by *aspital* contains unintended shift assignments, they can be manually modified to specific shifts. Alternatively, users can specify lists of desired and undesired shifts, which are implemented by adding the `pos_request` and `neg_request` predicates as facts. Since *aspital* is provided as a web service, all modifications can be made directly through a web browser (see Fig. 2 (a)). Additionally, shift assignments can be fixed, ensuring they remain unchanged in subsequent optimization iterations. For example, manually modified shifts can be fixed, or past shifts can be preserved during rescheduling. Conversely, parts of the schedule can be cleared and reconstructed as needed (see Fig. 2 (b) for an example where the first week's assignments are fixed while some assignments in the third week are cleared).

A key feature of *aspital* is its handling of non-modifiable parts of the schedule. If these parts do not pose any issues, preserving the existing schedule as much as possible is preferable. As reviewing a schedule can be burdensome, minimizing unnecessary changes reduces this effort. Additionally, in rescheduling scenarios, significant modifications to shift assignments may require confirmation from nurses, making it preferable to maintain the current schedule whenever possible. This problem, known as the *Minimal Perturbation Problem* (MPP, hereafter referred to as MP), aims to make minimal changes to an existing solution while adapting to new constraints or environmental changes. Traditionally, MP has been studied within CSPs [9, 3], but its relevance has recently received more attention [2, 12]. A common approach to MP is incorporating a minimal perturbation term into the objective function to keep deviations from the initial solution as small as possible. In this approach, the trade-off between resolving constraint violations and minimizing solution changes must be carefully adjusted by tuning the priority of perturbation minimization or, if incorporated into a single objective function, by adjusting its weight.

In this study, we apply the *Large Neighborhood Prioritized Search* (LNPS; [10]) method to generate modified schedules. As demonstrated by the experimental results in the next section, LNPS achieves faster solution improvement than MP in our NSP setting, making it more suitable for interactive use. LNPS was originally proposed to efficiently solve combinatorial optimization problems in ASP. It extends the *Large Neighborhood Search* (LNS) framework by replacing fixed solution segments with priority-based search, enabling a more flexible and efficient exploration of feasible solutions. In standard LNS, fixed

```
1 #heuristic ext_assigned(N,D,S) : prioritized(ext_assigned(N,D,S)). [1,true]
2 :- fixed(ext_assigned(N,D,S1)), ext_assigned(N,D,S2), S1 != S2.
```

Listing 3: ASP Encoding for Prioritized Search

segments heavily influence the choice of destruction operators, making it difficult to guarantee optimality. In contrast, LNPS allows prioritized search over all variable assignments, facilitating a broader and more effective search. Prioritized search refers to a method that controls the search process by preferentially selecting specific value assignments. In LNPS, the value assignments of the non-disrupted part of the solution are prioritized. However, this prioritization does not mean the assignments are fixed, as in LNS; if necessary to satisfy constraints, alternative values may be selected. In other words, LNPS makes certain values more likely to be chosen but does not enforce that they are always assigned. This prioritized search is implemented using the `#heuristic` statement in the ASP solver *clingo*, which is used to set search priorities.

We utilize LNPS for interactive nurse schedule modifications. However, our approach differs from standard LNPS in the following aspects: (1) LNPS periodically restarts the search process. While our method also performs periodic restarts, it additionally allows the head nurse to manually pause the search, apply modifications, and then restart it. We refer to the former as *automatic* restart and the latter as *manual* restart. Since *aspital* updates and displays solutions immediately on the web interface, the head nurse can monitor progress and pause the search when the solution stabilizes. (2) In LNPS, an appropriate destruction operator must be designed for each optimization problem. This operator is executed at every automatic restart, repeatedly performing destruction and reconstruction to explore solutions. In our approach, the head nurse manually selects the parts to be destroyed during a manual restart, which corresponds to clearing certain shift assignments. No destruction is performed during an automatic restart.[6] (3) As in LNS, part of the solution can be fixed.

During a manual restart, parts of the solution that are neither fixed nor cleared by the head nurse become the focus of prioritized search. However, if an automatic restart occurs in a subsequent search, priorities are reassigned based on the current variable assignments, and the search resumes accordingly. Thus, minimal perturbation is not guaranteed, but solution optimality is maintained.

Listing 3 illustrates the ASP encoding used to implement prioritized search. In both automatic and manual restarts, *aspital* assigns priorities to shift assignments by adding the fact `prioritized(ext_assigned(N,D,S))` and designates fixed shift allocations using the fact `fixed(ext_assigned(N,D,S))`.[7] The first line in Listing 3 specifies that if the fact `prioritized(ext_assigned(N,D,S))` exists, then `ext_assigned(N,D,S)` is heuristically prioritized and assigned as true. This is achieved using the `#heuristic` directive in the ASP solver *clingo*.[8] The directive `[1,true]` assigns a priority of `1` to the positive literal `ext_assigned(N,D,S)`. Since the default priority is `0`, the solver prefers assigning this literal to true over others with default priority. The second line enforces an integrity constraint that prevents modifications to fixed shifts.

[6] Destruction could be incorporated into automatic restarts, but an effective destruction operator has not yet been designed for our NSP model.

[7] Since these predicates must be asserted and retracted at each restart, *aspital* utilizes the multi-shot ASP solving of *clingo* to manage them dynamically. For details, see [10].

[8] To enable heuristic-driven search in *clingo*, the option `-heuristic=Domain` must be specified.

4 Experiments

In this section, we compare LNPS and MP using representative NSP instances from our hospital to demonstrate that LNPS is a suitable search strategy for schedule adjustments.

The evaluation instances cover a 28-day scheduling period with nurse counts of 10, 20, 30, 40, and 50. For each setting, 10 instances were generated using different random seeds, totaling 50 instances. The objective function follows a four-tier priority structure: (1) staff preferences, (2) inter-shift constraints to prevent invalid assignments, (3) daily staffing and shift frequency constraints, and (4) all remaining constraints with the lowest priority. Requested shifts, both desired and undesired shifts, appear in 10% of the schedule cells and are treated as hard constraints. As a result, 60% of instances are initially unsatisfiable, requiring relaxation. In such cases, hard constraints are softened with a priority higher than that of the original soft constraints, while maintaining the same hierarchical order. These instances reflect the typical NSP in our hospital. The instance data and instance generation script are publicly available in our repository.

To evaluate schedule reconstruction performance, an initial solution was generated for each instance within a one-hour time limit. Subsequently, 5% additional requested shifts were introduced, and evaluation was conducted under three modification scenarios:

1. Entire set reconstructed: The initial solution is completely discarded, and a new schedule is generated from scratch. This serves as a baseline for comparing search strategies without relying on prior assignments.
2. First half retained, second half reconstructed: The first half of the initial solution is preserved, while the second half is cleared and reconstructed.
3. Entire set retained: The entire initial solution is retained.

In scenarios 2 and 3, requested shifts are added even to the retained portions, potentially causing constraint violations, which may require modifications to the initial solution to ensure feasibility.

In the evaluation experiments, we compared three search strategies: LNPS, MP, and MP with Initial-value Search (MP+IS). LNPS employs a restart mechanism where, if no improved solution is found within a given time interval t ($t \in \{10, 30, 60\}$), the search is automatically restarted. Each configuration is denoted as LNPS-t based on the selected time threshold. MP minimizes modifications to the initial solution by introducing a penalty term into the objective function:

```
#minimize { 1@mp_priority,N,D,S :
    prioritized(ext_assigned(N,D,S)), not ext_assigned(N,D,S) }.
```

where `mp_priority` determines the priority of this objective function. In our experiments, we set it to the highest, middle, and lowest priority levels, corresponding to MP-High, MP-Mid, and MP-Low, respectively. MP+IS extends MP by incorporating initial-value-based search, where retained assignments from the initial solution guide the search. These retained values are assigned using *clingo*'s `#heuristic` directive, as shown in Listing 4. In this encoding, `[10, init]` increases the likelihood of selecting `ext_assigned(N,D,S)` in the early search stages.[9] The second line, `[1, sign]`, instructs the solver to first attempt assigning the variable to true once selected as a decision variable. Similar to MP, we define three variations: MP-IS-High, MP-IS-Mid, and MP-IS-Low.

The performance of each search strategy was evaluated under time limits of 60 and 3600 seconds. The 60-second limit represents a real-time usage scenario, while the 3600-second limit corresponds to

[9]The value 10 corresponds to the default initial score in the VSIDS heuristic. Preliminary tests with values 1, 10, and 100 showed no significant differences.

```
#heuristic ext_assigned(N,D,S) : prioritized(ext_assigned(N,D,S)). [10,init]
#heuristic ext_assigned(N,D,S) : prioritized(ext_assigned(N,D,S)). [1,sign]
```

Listing 4: ASP Encoding for Heuristic Assignment Specification

final schedule generation. All experiments were conducted on a system running Red Hat Enterprise Linux 8 with an Intel Xeon Platinum 8480+ processor and 512 GiB of RAM. The solver script was written in Python and executed *clingo* 5.7.1 via its Python API, using the trendy search strategy optimized for industrial instances. The solver ran in single-threaded mode, and each instance was tested three times. Additional experimental results, including extended scenarios, alternative search strategies, and preliminary experiments, are provided in our repository's `comparison` directory.

Fig. 3 shows the results of the entire shift reconstruction scenario as a cumulative performance plot (cactus plot). Each search strategy's results are plotted in ascending order of the objective function value. Since our NSP is a prioritized multi-objective optimization problem, we apply scalarization for visualization. The aggregated objective value is computed as follows:

$$F = \sum_{i=1}^{n} w_i f_i', \quad f_i' = \frac{f_i - \min f_i}{\max f_i - \min f_i}, \quad w_i = \beta^{(n-i)}$$

where n is the number of objective functions, and i represents the priority ranking. Here, f_i' is the min-max normalized objective value, ensuring comparability across different scales. The weight parameter β is set to 10 to strongly prioritize higher-ranked objectives while preserving the influence of lower-priority ones.

When reconstructing the entire shift schedule (Fig. 3), all search strategies perform similarly, indicating that there is little difference in schedule generation from scratch. However, when part of the initial solution is retained, performance varies across strategies. Figs. 4 and 5 show the results for 60-second and 3600-second time limits, respectively.

As shown in Fig. 4, LNPS improves the objective function faster while making fewer modifications to the initial solution. This advantage stems from LNPS avoiding additional constraints for minimal perturbations, thereby reducing computational overhead. However, with longer computation times (Fig. 5), LNPS exhibits an increasing number of modifications, eventually reaching levels similar to MP+IS-Low and MP+IS-Mid. This occurs because LNPS automatically restarts the search using the latest assignment as a new initial solution, causing modifications to accumulate over time. Nonetheless, this mechanism helps escape local optima, potentially leading to better solutions. Fig. 5 further shows that LNPS achieves objective function values comparable to MP-Low and MP-Mid while maintaining a lower modification rate. MP-Low and MP-Mid achieve better objective function values but require more modifications. In contrast, MP-IS-High reduces modifications but sacrifices objective function quality. LNPS balances both, optimizing the objective function while minimizing modifications, making it well-suited for interactive use due to its faster computation time.

5 Discussion

Our nurse scheduling system at the University of Yamanashi Hospital builds upon established research in ASP [5, 2], CP [11, 1], and SAT [8, 13]. Going beyond these foundations, our field experience revealed specific challenges in the real-world application, and we outlined effective strategies for their resolution. To conclude, let us present some feedback from nurses using *aspital*. Positive feedback includes comments

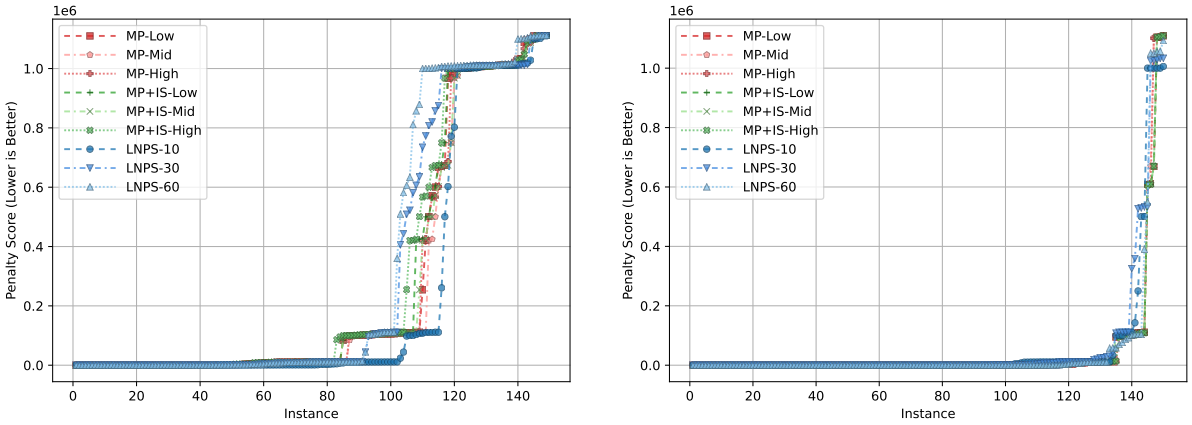
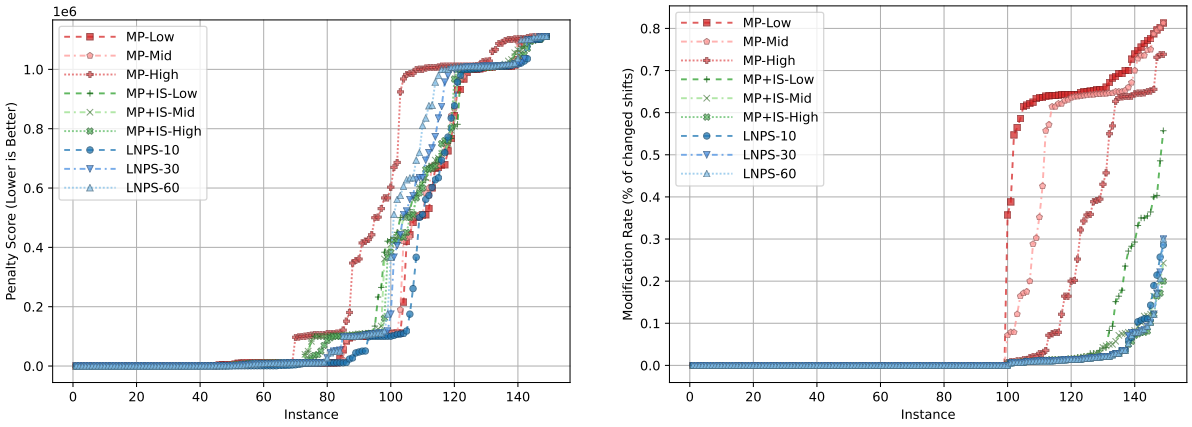
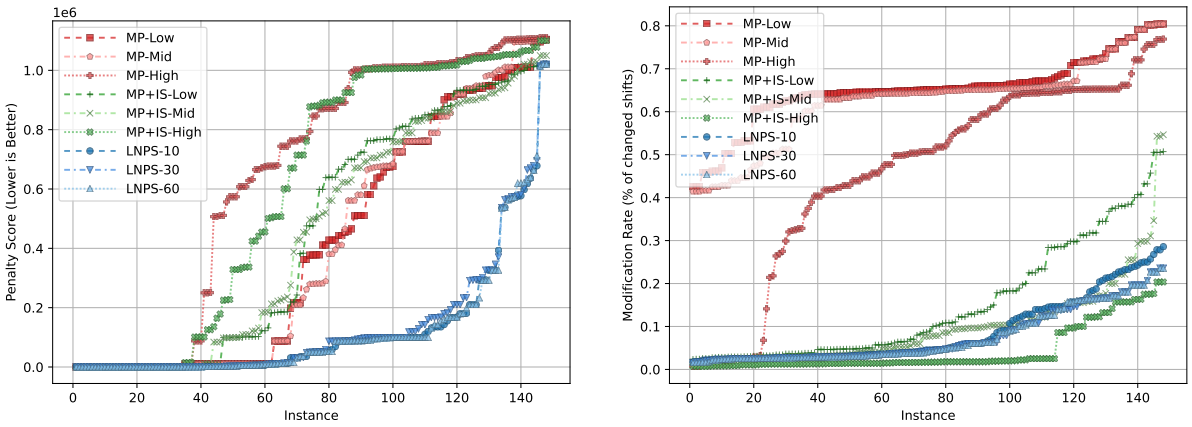


Figure 3: Entire set reconstructed (Left: 60-second limit, Right: 3600-second limit)

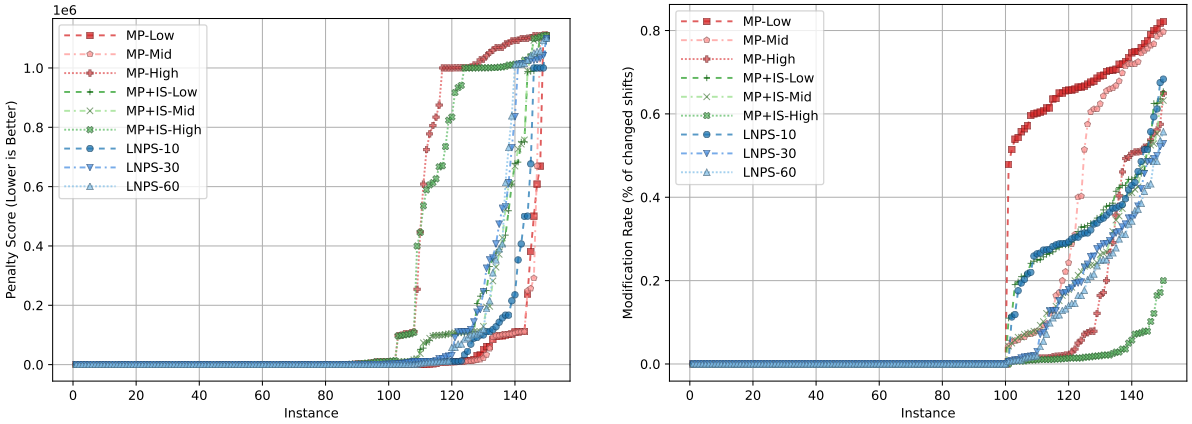


(a) First half retained, second half reconstructed (Left: Penalty score, Right: Modification rate)

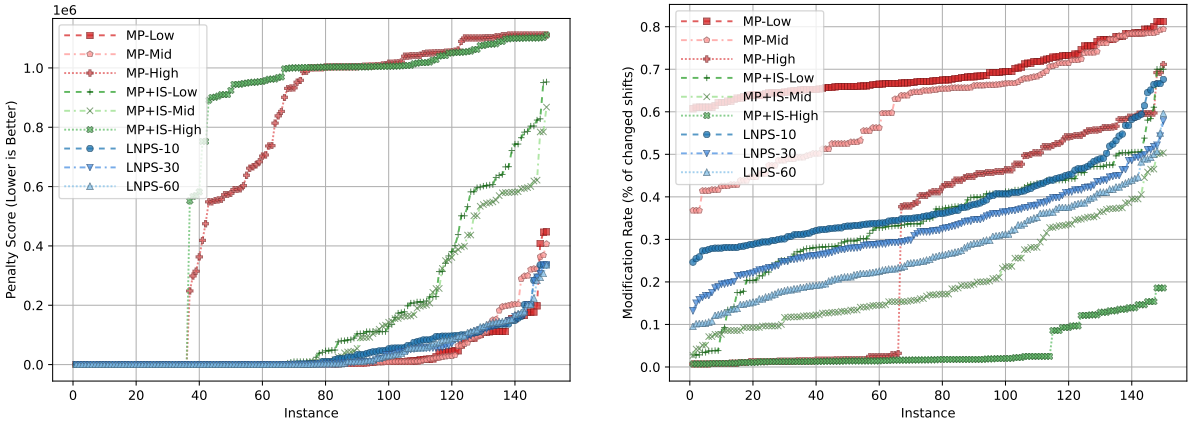


(b) Entire set retained (Left: Penalty score, Right: Modification rate)

Figure 4: Comparison of search strategies under a 60-second time limit



(a) First half retained, second half reconstructed (Left: Penalty score, Right: Modification rate)



(b) Entire set retained (Left: Penalty score, Right: Modification rate)

Figure 5: Comparison of search strategies under a 3600-second time limit

from head nurses indicating a reduced workload in shift scheduling. Additionally, nurses reported an increased opportunity to work with a diverse range of colleagues and found it easier to submit leave requests without concern for the head nurse's burden of adjusting schedules. On the other hand, most negative feedback pertained to shortcomings in the modeling. For instance, some nurses requested the avoidance of particularly demanding shift patterns, while others suggested a more balanced allocation of consecutive days off and night shifts. In response to such feedback, improvements have been made, such as assigning weekly rest days adjacent to requested leave (S_8) and introducing constraints to equalize workload distribution (S_9). Regularly incorporating feedback from both head nurses and staff nurses remains essential for further refining the modeling and enhancing the system's effectiveness.

References

[1] S. Abdennadher & H. Schlenker (1999): *Nurse Scheduling using Constraint Logic Programming*. In J. Hendler & D. Subramanian, editors: *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, AAAI/MIT Press, pp. 838–843. Available at `http://www.aaai.org/Library/IAAI/1999/iaai99-118.php`.

[2] M. Alviano, C. Dodaro & M. Maratea (2018): *Nurse (re)scheduling via answer set programming*. *Intelligenza Artificiale* 12(2), pp. 109–124, doi:10.3233/IA-170030.

[3] R. Barták, T. Müller & H. Rudová (2003): *A New Approach to Modeling and Solving Minimal Perturbation Problems*. In K. R. Apt, F. Fages, F. Rossi, P. Szeredi & J. Váncza, editors: *Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2003*, *Lecture Notes in Computer Science* 3010, Springer, pp. 233–249, doi:10.1007/978-3-540-24662-6_13.

[4] S. Ceschia, N. Dang, P. De Causmaecker, S. Haspeslagh & A. Schaerf (2019): *The Second International Nurse Rostering Competition*. *Annals of Operations Research* 274(1-2), pp. 171–186, doi:10.1007/S10479-018-2816-0.

[5] C. Dodaro & M. Maratea (2017): *Nurse Scheduling via Answer Set Programming*. In M. Balduccini & T. Janhunen, editors: *Proceedings of the Fourteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, *Lecture Notes in Artificial Intelligence* 10377, Springer-Verlag, pp. 301–307, doi:10.1007/978-3-319-61660-5_27.

[6] A. Ikegami (2005): *Nurse Scheduling: Survey, Modeling, and Algorithms (in Japanese)*. *Proceedings of the Institute of Statistical Mathematics* 53(2), pp. 231–259. Available at `https://cir.nii.ac.jp/crid/1050001202804340224`.

[7] R. Kaminski, J. Romero, T. Schaub & P. Wanko (2023): *How to Build Your Own ASP-based System?!* *Theory and Practice of Logic Programming* 23(1), pp. 299–361, doi:10.1017/S1471068421000508.

[8] S. Kundu & S. Acharyya (2008): *A SAT Approach for Solving the Nurse Scheduling Problem*. In: *TENCON 2008 - 2008 IEEE Region 10 Conference*, pp. 1–6, doi:10.1109/TENCON.2008.4766380.

[9] H. El Sakkout, T. Richards & M. Wallace (1998): *Minimal Perturbance in Dynamic Scheduling*. In H. Prade, editor: *Proceedings of the Thirteenth European Conference on Artificial Intelligence (ECAI'98)*, John Wiley & sons, pp. 504–508.

[10] I. Sugimori, K. Inoue, H. Nabeshima, T. Schaub, T. Soh, N. Tamura & M. Banbara (2024): *Large Neighborhood Prioritized Search for Combinatorial Optimization with Answer Set Programming*. In P. Marquis, M. Ortiz & M. Pagnucco, editors: *Proceedings of the Twenty-first International Conference on Principles of Knowledge Representation and Reasoning (KR'24)*, doi:10.24963/KR.2024/72.

[11] G. Weil, K. Heus, P. Francois & M. Poujade (1995): *Constraint Programming for Nurse Scheduling*. *IEEE Engineering in Medicine and Biology Magazine* 14(4), pp. 417–422, doi:10.1109/51.395324.

[12] T. I. Wickert, P. Smet & G. V. Berghe (2019): *The Nurse Rerostering Problem: Strategies for Reconstructing Disrupted Schedules*. Computers & Operations Research 104, pp. 319–337, doi:10.1016/j.cor.2018.12.014.

[13] Tsung-Shun Wu, Chia-Ling Ho & Chien-Liang Chen (2015): *A SAT Approach to Nursing Scheduling Problem*. In James J. (Jong Hyuk) Park, Yi Pan, Cheonshik Kim & Yun Yang, editors: Future Information Technology - II, Springer Netherlands, Dordrecht, pp. 37–43, doi:10.1007/978-94-017-9558-6_5.

Hybrid MKNF for Aeronautics Applications: Usage and Heuristics

Arun Raveendran Nair Sheela
Université Clermont Auvergne, LIMOS
Thales

Florence De Grancey
Thales

Christophe Rey
Université Clermont Auvergne, LIMOS
CNRS, France

Victor Charpenay
École des Mines de Saint-Étienne, LIMOS
CNRS, France

The deployment of knowledge representation and reasoning technologies in aeronautics applications presents two main challenges: achieving sufficient expressivity to capture complex domain knowledge, and executing reasoning tasks efficiently while minimizing memory usage and computational overhead. An effective strategy for attaining necessary expressivity involves integrating two fundamental KR concepts: rules and ontologies. This study adopts the well-established KR language Hybrid MKNF owing to its seamless integration of rules and ontologies through its semantics and query answering capabilities. We evaluated Hybrid MKNF to assess its suitability in the aeronautics domain through a concrete case study. We identified additional expressivity features that are crucial for developing aeronautics applications and proposed a set of heuristics to support their integration into Hybrid MKNF framework.

1 Introduction

The aeronautics industry is acknowledged as a safety-critical domain where system failures can have serious safety implications for stakeholders. Therefore, any software system integrated into this sector must exhibit trustworthy attributes such as reliability, robustness, and explainability to prevent such consequences. Moreover, many software systems in this field operate on resource-constrained hardware such as microcontrollers. In this context, Knowledge Representation and Reasoning (KRR) shows significant promise as it enables systems to make transparent and explainable decisions. With the assistance of domain experts, knowledge specific to aeronautics applications has been captured using formal representation languages, such as ontologies. This knowledge encompasses critical operational information and can be combined with real-time data to facilitate timely decision-making through logical reasoning. However, adapting KRR to the aeronautics domain presents several significant challenges. These include ensuring sufficient expressivity to encapsulate all complex domain knowledge and effectively executing the necessary reasoning tasks with limited computational resources and minimal latency.

An effective approach to addressing the expressivity challenge involves integrating rules and ontologies, which is a well-researched area. Ontologies, typically based on Description Logics (DLs), operate under the open-world assumption (OWA), meaning that the absence of information does not equate to its negation. They encapsulate general domain knowledge through concepts and their relationships [5]. In contrast, Logic Programming (LP) rules function under the closed-world assumption (CWA), assuming that what is not known to be true is considered false. These rules represent knowledge through if-then relationships [42]. Owing to their complementary features, combining DLs and LP rules enhances the expressiveness of the resulting KRR language, which is known as a hybrid knowledge base. Various

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 349–366, doi:10.4204/EPTCS.439.24

© A.R.N. Sheela, F. De Grancey, C. Rey & V. Charpenay
This work is licensed under the
Creative Commons Attribution License.

methodologies have been proposed for creating hybrid knowledge bases, each offering unique advantages and trade-offs [28], [10], [13], [37]. In this study, we explore Hybrid MKNF language, which provides a unified logic that integrates the semantics of DLs and LP [38]. Both well-founded semantics [29] and answer-set semantics [39] have been proposed for Hybrid MKNF. Given that complex query answering constitutes the primary reasoning task within the aeronautics domain and that well-founded semantics offers greater computational efficiency for handling it, we adopted this semantics in our approach.

The primary contribution of this study lies in exploring Hybrid MKNF under well-founded semantics and its implementation in the aeronautics sector, along with its practical adaptability through a comprehensive evaluation. To assess this, we conducted a series of experiments focusing on preprocessing delays, query answering performance, and resource utilization. We also acknowledge that Hybrid MKNF under well-founded semantics does not inherently support all expressive features required for modeling complex aviation scenarios. To address this limitation, we define a set of core expressive needs using a semi-realistic use case, and propose heuristics to demonstrate how these features can be effectively integrated into Hybrid MKNF framework. In addition, we address future research directions to further enhance the features of Hybrid MKNF in an aeronautics context.

Section 2 presents a semi-realistic use case within the aeronautics sector to illustrate the concepts discussed in this study. Section 3 provides an overview of Hybrid MKNF framework, introduces the NoHr reasoner, and explains its efficient usage in this context. Section 4 proposes heuristics to enhance expressiveness with classical negation and integrity constraints. Section 5 describes the implementation of these features. Section 6 reviews related work and justifies the choice of Hybrid MKNF. Finally, Section 7 concludes the study and outlines future directions.

2 Use Case

To address a use case in the aeronautics domain requiring the integration of DLs and LP rules, we propose the NOTAM-Aware Reasoning System (NARS). Before departure, pilots are tasked with manually reviewing all Notice to Airmen (NOTAMs) messages, which are official notifications issued to communicate time-sensitive changes in airspace status, environmental conditions, runway closures, and other relevant factors. This review aims to ensure that all NOTAMs are consistent with flight plans. Below is an example of a preprocessed NOTAM and flight plan:

```
Notam001 :hasaffectedBy rw1;                Flight:AF123 :hasDepartureAirport LFBO ;
         :hasRunwayStatus :Closed;                       :hasDestinationAirport LFPG ;
         :hasAirport LFPG;                               :hasDepartureTime "2025-04-18T06:00:00Z";
         :haStartTime "2025-04-18T06:00:00Z";            :hasArrivalTime "2025-04-18T07:30:00Z";
         :hasValidEndTime "2025-04-18T12:00:00Z";        :landingRunway rw1 ;
         :hasIssuedBy :ATC_Agency1 .                     :hasAlternateAirport ex:LFOB .
```

For example, if a NOTAM indicates that the designated runway is unavailable at the flight's scheduled arrival time, the pilot must either request a revised flight plan or delay the flight. Manually analyzing each NOTAM to extract such information and making appropriate decisions, whether by the pilot or supporting staff, is both time-consuming and prone to error. To address this challenge, we propose NARS that assists pilots by automatically providing recommendations based on relevant NOTAMs. Some of the key operations performed by NARS include the following:

1. Decisions should be guided solely by valid NOTAMs. Any NOTAM that is invalid or contradictory, due to missing critical data or conflicting information from different authorities, must be disregarded, and pilots should be notified of its invalidity.

2. If the current alternate airports are found to be unavailable due to constraints reported in NOTAMs, it is necessary to recommend the designation of new alternate airports.
3. In emergency landing situations, nearby airports are suggested along with runways that meet the necessary parameters for safely landing the aircraft. One crucial parameter is the runway length, which must exceed the aircraft's required landing distance.

The knowledge base for NARS can be constructed using Hybrid MKNF. On the DL side, static and hierarchical domain knowledge, such as airport infrastructure, different types of runways, and airspace zones can be represented. This structured knowledge provides a formal vocabulary to describe entities and their relationships. For example, stating that "*a runway is part of an airport*" or "*runway has an instrument landing system (ILS)*". In contrast, LP handles dynamic knowledge, which is encoded as conditional if-then rules. For instance, a rule might specify that "*if a runway is closed during the scheduled landing time, then an alternate runway should be proposed*" or "*if critical equipment at an airport is out of service, the flight operations team must be notified*". Specific instances, such as individual NOTAMs or flight plan, can be formalized as assertions.

3 Hybrid MKNF

Hybrid MKNF, as introduced in [38], is based on the logic of Minimal Knowledge and Negation-as-a-Failure (MKNF), an extension of first-order logic that incorporates two modal operators, namely **K** and **not** originally described in [34]. The **K** operator denotes knowledge known by the system, whereas the **not** operator signifies knowledge not known by the system, functioning analogously to default negation in LP and facilitating closed-world reasoning. A Hybrid MKNF under well-founded semantics is defined as $KB = (O,P)$ where O is a DL ontology and P is the set of MKNF rules of the form [30]:

$$\mathbf{K}\, H \leftarrow \mathbf{K}\, A_1, \ldots, \mathbf{K}\, A_m, \mathbf{not}\, B_1, \ldots, \mathbf{not}\, B_n. \tag{1}$$

where H, A_i for $1 \leq i \leq m$, and B_j for $1 \leq j \leq n$ are first-order atoms. The rule (1) can be read as "*if all A_i are known to hold and all B_j are not known to hold then H is known to hold*". We consider a simplified scenario from the use case, in which airport and runway availability is inferred from NOTAM data, and operational flight recommendations are derived using a Hybrid MKNF knowledge base, $KB = (O,P)$:

$$\text{Airport} \sqcap \forall \text{hasRwy.CldRwy} \sqsubseteq \text{CldAirport} \tag{2}$$

$$\text{Airport} \sqcap \exists \text{hasRwy.OpnRwy} \sqsubseteq \text{OpnAirport} \tag{3}$$

$$\text{OpnRwy} \sqsubseteq \text{Rwy} \tag{4}$$

$$\text{CldRwy} \sqcap \text{OpnRwy} \sqsubseteq \bot \tag{5}$$

$$\mathbf{K}\ \text{CldRwy}(X) \leftarrow \mathbf{K}\ \text{NOTAM}(X), \mathbf{K}\ \text{affdBy}(X,Y), \mathbf{K}\ \text{Rwy}(Y),\ \mathbf{K}\ \text{opStat}(X, \text{closed}). \tag{6}$$

$$\mathbf{K}\ \text{OpnRwy}(X) \leftarrow \mathbf{K}\ \text{Rwy}(X),\ \mathbf{not}\ \text{CldRwy}(X). \tag{7}$$

$$\mathbf{K}\ \text{recommendDelay}(X) \leftarrow \mathbf{K}\ \text{Flight}(X), \mathbf{K}\ \text{hasDepartureAirport}(X,Y), \mathbf{not}\ \text{OpnAirport}(Y). \tag{8}$$

Rule (7) infers that *rwy(rw1)* is considered an *OpnRwy(rw1)* by default, unless there is explicit evidence indicating that *rwy(rw1)* is a *CldRwy(rw1)*. The predicate *CldRwy(rw1)* can be derived when there exists a NOTAM that affects the specific runway, as described in rule (6). This can be expressed using the **not** operator: **not** *CldRwy(X)* signifies that it is not known that a runway is closed. This allows the inference that, in the absence of a NOTAM message indicating a closure, the runway remains operational. Furthermore, by asserting the following additional facts: *hasRwy(lfbo, rw1)* and *Airport(lfbo)* and applying rule (7) in conjunction with DL axiom (3), it can be concluded that *Airport(lfbo)* qualifies as an *OpnAirport*.

The semantics of Hybrid MKNF can be formalized by transforming the knowledge base into a first-order formula with modal operators [30]. Consider $KB_1 = (O_1, P_1)$ where O_1 contain DL axiom (4) and an assertion *Rwy(rw1)*, and P_1 contain a rule (7). The first-order translation of KB_1, denoted as $\pi(KB_1)$ is given by $\mathbf{K}\pi(O_1) \wedge \pi(P_1)$:

$$\forall X(\mathbf{K}\ \mathrm{Rwy}(X) \leftarrow \mathbf{K}\ \mathrm{OpnRwy}(X)). \tag{9}$$

$$\forall X(\mathbf{K}\ \mathrm{OpnRwy}(X) \leftarrow \mathbf{K}\ \mathrm{Rwy}(X),\ \mathbf{not}\ \mathrm{CldRwy}(X)). \tag{10}$$

A model of an MKNF formula is defined as a maximal set of interpretations that satisfy the formula, such that no proper superset of this set also satisfies the formula [34]. KB_1 is satisfiable if there exists a MKNF model for $\pi(KB_1)$ and KB_1 entails a first-order formula ψ, denoted as $KB_1 \models \psi$ if and only if $\pi(KB_1) \models_{MKNF} \psi$. The semantics of $\pi(KB_1)$ can be explained using either answer set [39] or well-founded semantics [30]. In the field of aeronautics, well-founded semantics is essential because they establish a single three-valued model comprising *true*, *false*, and *undefined* atoms, thereby facilitating an efficient top-down query answering. To explain well-founded semantics for Hybrid MKNF, a three-valued MKNF interpretation is defined as a pair (M,N), where both M and N are sets of first-order interpretations with $N \subseteq M$. To avoid unintended consequences when integrating DL with LP, the standard name assumption (SNA) is enforced, ensuring each constant uniquely identifies an individual and equality is explicitly handled as a congruence relation [39].

Truth evaluation of first-order atoms with modal operators in relation to the three-valued MKNF model, proceeds as follows. An atomic formula $P(t_1,\ldots,t_n)$ is evaluated as *true* under a first-order interpretation I if the tuple $(t_1^I,\ldots,t_n^I)$ is an element of the interpretation of the predicate P in I, denoted P^I; formally, if $(t_1^I,\ldots,t_n^I) \in P^I$. Otherwise, the formula is evaluated as *false* when $(t_1^I,\ldots,t_n^I) \notin P^I$. The implication $\varphi_1 \supset \varphi_2$ is evaluated as *true* if the truth value of φ_2 is greater than or equal to that of φ_1 with respect to the order $f < u < t$, and *false* otherwise. The truth value of $\mathbf{K}\ \varphi$ is *true* if φ evaluates to true in all interpretations in M, *false* if φ evaluates to false in some interpretation in $N \subseteq M$, and *undefined* otherwise. Similarly, **not** φ is *true* if φ is false in some interpretation in $N \subseteq M$, *false* if φ is true in all interpretations in M, and *undefined* otherwise [30].

An MKNF interpretation pair (M,N) satisfies a closed MKNF formula φ, denoted $(M,N) \models \varphi$, if and only if $(M,N)(\varphi) = t$, meaning that all rules within φ evaluate to true under this interpretation. An MKNF interpretation pair (M,N) is considered a three-valued MKNF model for a formula φ if it satisfies two key conditions. First, it must satisfy the formula φ itself. Second, it must be minimal in the sense that one cannot extend M or N to larger interpretation pairs that still satisfy φ. The only three-valued model of KB_1: *M = {{Rwy(rw1), OpnRwy(rw1)}, {Rwy(rw1), OpnRwy(rw1), CldRwy(rw1)}}, N = {{Rwy(rw1), OpnRwy(rw1)}, {Rwy(rw1), OpnRwy(rw1), CldRwy(rw1)}}.*

Query Answering: The alternating fixed-point method provides a bottom-up strategy for calculating the three-valued, well-founded model of a Hybrid MKNF knowledge base [30]. Initially, this model must be computed using the proposed bottom-up approach. Once computed, it can be stored in a database for querying purposes. However, this method is impractical for large knowledge bases, particularly when only specific parts of information are relevant to answering a query. To address this issue, a query answering algorithm called SLG(O) resolution has been proposed, which is both sound and complete with respect to the well-founded Hybrid MKNF model. SLG(O) resolution is a goal-directed query answering that employs SLG resolution to manage LP rules, where O acts as the oracle responsible for handling queries related to the DL component [1]. The resolution process begins with a query that is matched against the rule head, subsequently generating new subgoals derived from the body of that rule. When a subgoal corresponds to a DL atom, the system invokes the DL oracle to determine whether

the atom is entailed by DL. Throughout the process, tabling is used to store the intermediate results, thereby preventing infinite loops and ensuring termination. Before implementing SLG(O) resolution, a knowledge base doubling step must be conducted to align the algorithm with the bottom-up computation of the well-founded MKNF model [1].

Consider $KB = (O, P)$ to be a Hybrid MKNF knowledge base and in [1] introduce new predicates L^d and NL for each predicate L appearing in KB, and then define:

1. O^d by substituting each concepts or role L in O by L^d; and
2. P^d by transforming each rule of form (1) occurring in P into two rules:

   i $\mathbf{K}\, H \leftarrow \mathbf{K}\, A_1, \ldots, \mathbf{K}\, A_n, \mathbf{not}\, B_1^d, \ldots, \mathbf{not}\, B_m^d$ and either,

   ii(a) $\mathbf{K}\, H^d \leftarrow \mathbf{K}\, A_1^d, \ldots, \mathbf{K}\, A_n^d, \mathbf{not}\, B_1, \ldots, \mathbf{not}\, B_m, \mathbf{not}\, NH$ if H is a DL concept or role; or

   ii(b) $\mathbf{K}\, H^d \leftarrow \mathbf{K}\, A_1^d, \ldots, \mathbf{K}\, A_n^d, \mathbf{not}\, B_1, \ldots, \mathbf{not}\, B_m$ otherwise.

The doubled Hybrid MKNF knowledge base is defined as $KB^d = (O, O^d, P^d)$. In this transformation, each original predicate L in the knowledge base KB is paired with two predicates, L and L^d, where L^d indicates the non-falsity of L. In addition, a new predicate NL, semantically equivalent to the classical negation $\neg L$, is introduced. This predicate prevents the derivation of L^d when $\neg L$ is entailed in the ontology. To enforce this, **not** NL is added to the body of every rule, with L^d as its head, ensuring that L^d can only be inferred when $\neg L$ is not supported by the ontology. This transformation process is called a semi-negative transformation. It aids in verifying MKNF consistency by identifying cases where L is true but L^d is not. In such instances, because L is true and L^d is false, it follows that $\neg L$ is entailed by the DL ontology O, leading to inconsistency.

3.1 NoHr Reasoner

The *NoHr* reasoner employs SLG(O) resolution, fundamentally translating the doubled Hybrid MKNF knowledge base into an equivalent set of rules [27]. This method facilitates efficient reasoning through query answering by using *XSB Prolog*. Specifically, the doubled Hybrid MKNF knowledge base, represented as $KB = (O, O^d, P^d)$, is converted into a semantically equivalent form: $KB = (\emptyset, P^d \cup P^O)$, where P^d is the doubled logic program, and P^O is the DL ontology translated into rules, followed by knowledge base doubling. The NoHr reasoner employs two distinct translation methodologies to convert DL into rules. First, a classification algorithm is used to deduce the subsumption relationships, which are then translated into LP rules [26]. For classification, NoHr integrates three existing DL reasoners: ELK, HermiT, and Konclude, with Hermit and Konclude supporting more expressive DL constructors. Second, for the QL profile, a direct translation method was applied to convert DL axioms into Logic Programs without prior classification [7]. We chose to focus on the classification-based translation with the HermiT reasoner, as it supports more expressive DL constructors than QL. The various steps involved in NoHr with the classification-based translator are illustrated in figure 1.

Consider DL axioms (4) and (5), which are translated into rules according to the NoHr proposition, followed by knowledge base doubling and the semi-negative transformation (Step 1 to Step 6 in figure 1, see [26] for more details):

$$\mathrm{Rwy}(X) \leftarrow \mathrm{OpnRwy}(X). \quad (11) \qquad \mathrm{NCldRwy}(X) \leftarrow \mathrm{OpnRwy}(X). \quad (13)$$

$$\mathrm{Rwy}^d(X) \leftarrow \mathrm{OpnRwy}^d(X), \mathbf{not}\, \mathrm{NRwy}(X). \quad (12) \qquad \mathrm{NOpnRwy}(X) \leftarrow \mathrm{CldRwy}(X). \quad (14)$$

The NoHr reasoner exhibits paraconsistent behavior through a combination of knowledge base doubling and semi-negative transformation. This approach allows reasoning to persist, even when parts of

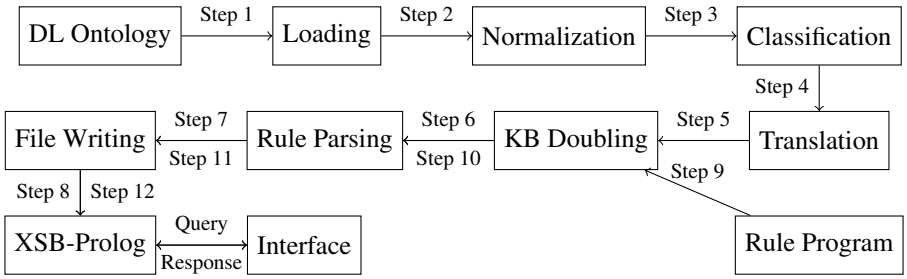


Figure 1: Different steps in NoHr Reasoner for EL Profile

the knowledge base are contradictory, by isolating and focusing on consistent fragments. In a Hybrid MKNF knowledge base, contradictions occur when both an atom H and its classical negation $\neg H$ are derivable. For example, consider adding the two assertions *OpnRwy(rw1)* and *cldRwy(rw1)*. According to DL axiom (5), these assertions are contradictory, as a runway cannot be both operational and closed simultaneously. After applying knowledge base doubling and the semi-negative transformation (Step 1 to Step 6 in figure 1), these assertions are translated as follows:

$$\mathrm{OpnRwy}(rw1). \quad (15) \qquad \mathrm{OpnRwy}^d(rw1) \leftarrow \mathbf{not}\ \mathrm{NOpnRwy}(rw1). \quad (17)$$

$$\mathrm{CldRwy}(rw1). \quad (16) \qquad \mathrm{CldRwy}^d(rw1) \leftarrow \mathbf{not}\ \mathrm{NCldRwy}(rw1). \quad (18)$$

Using rule (11), we can infer *Rwy(rw1)*. However, according to rule (12), we cannot infer $\mathrm{Rwy}^d(rw1)$ because *NOpnRwy(rw1)* holds, as established by rule (14). A literal H is considered *consistently true* only if both H and H^d are true. In this case, although $\mathrm{Rwy}(rw1)$ is true, $\mathrm{Rwy}^d(rw1)$ is false. Thus, the situation is treated as an inconsistency. Consequently, $\mathrm{Rwy}(rw1)$ is excluded from the set of consistent query answers. By employing this syntactic approach, the NoHr reasoner effectively detects contradictions, ensuring that only consistent information is conveyed in response to queries, wherein an atom and its duplicate are considered true.

Below, we outline the consistency-resolving techniques employed by the NoHr reasoner. Consider a conjunctive query of the form:

$$q(X) \leftarrow A_1, A_2, \ldots, A_n, \mathbf{not}\ B_1, \mathbf{not}\ B_2, \ldots, \mathbf{not}\ B_m \quad (19)$$

where $A_1, A_2, \ldots, A_n, \mathbf{not}\ B_1, \mathbf{not}\ B_2, \ldots, \mathbf{not}\ B_m$ represent the conjunction of predicates being queried within the context of a Hybrid MKNF framework. To address the paraconsistent feature, the query is transformed into two distinct queries:

$$q(X) \leftarrow A_1, A_2, \ldots, A_n, \mathbf{not}\ B_1^d, \mathbf{not}\ B_2^d, \ldots, \mathbf{not}\ B_m^d \quad (20)$$

$$q^d(X) \leftarrow A_1^d, A_2^d, \ldots, A_n^d, \mathbf{not}\ B_1, \mathbf{not}\ B_2, \ldots, \mathbf{not}\ B_m \quad (21)$$

Consistent answers were identified by querying the Prolog engine with: $q(X) \wedge q^d(X)$ while contradictory or inconsistent answers can be determined using: $q(X) \wedge \mathbf{not}\ q^d(X)$. This approach enables the system to manage contradictions by clearly separating consistent conclusions from contradictory ones, ensuring that the presence of contradictions does not undermine the soundness of the reasoning process.

Evaluation: A comprehensive evaluation of the NoHr reasoner is essential for its integration into aeronautics applications, to assess its performance in real-world operational environments. This evaluation should include the identification of DL constructors supported by NoHr, which is particularly beneficial for knowledge engineers involved in ontology modeling. Additionally, it is essential to measure the

time required for each preprocessing phase of the DL ontology and logic programs. This determines the time required for the reasoner to become operational after new knowledge is introduced into either component or during a system reboot. Furthermore, examining the latency of query responses by the NoHr reasoner is crucial, as it must deliver results in less than a second due to the safety-critical nature of aeronautics, where delays in query answering can pose significant risks. Another important parameter is resource utilization; as previously mentioned, the reasoner may operate on constrained devices with limited resources. Therefore, the reasoner should not consume excessive memory or CPU resources.

To analyze the preprocessing delay of the DL component, we used the OWL2Bench benchmark, which is an extension of the LUBM dataset designed to generate DL ontologies of varying sizes across four OWL profiles: EL, QL, RL, and DL [43]. We used this benchmark to evaluate the DL part with the NoHr reasoner. For the logic program component, we employed the benchmark provided by the NoHr reasoner, which generates a number 'n' of rules based on an associated DL ontology. For each DL ontology generated from the OWL2Benchmark, utilizing the rule benchmark provided by the NoHr reasoner, rules are generated in quantities of 10, 100, 1,000, 10,000, and 25,000. The rule benchmark is available here: `https://github.com/NoHRReasoner/NoHR/tree/master/nohr-benchmark` and OWL2Bench is available here: `https://github.com/kracr/owl2bench`. For query evaluation, we used a set of queries provided with the LUBM benchmark, making minor modifications to match the structure and vocabulary of the DL ontologies generated by the OWL2Bench benchmark. These were then converted into equivalent queries that were compatible with the Prolog engine. Queries are available here for reference: `https://swat.cse.lehigh.edu/projects/lubm/queries-sparql.txt`. To evaluate the memory consumption and CPU utilization, we used ontology files containing varying numbers of axioms.

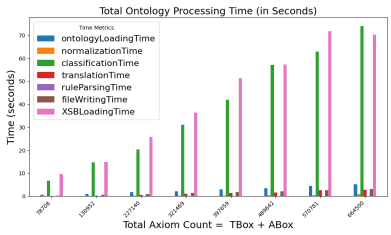


Figure 2: Ontology Processing

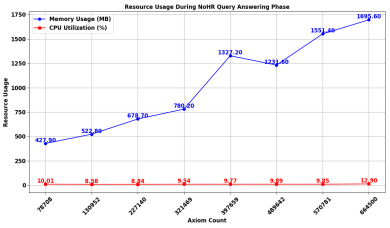


Figure 3: Resource consumption

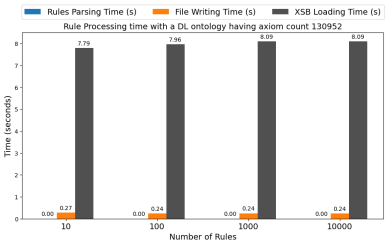


Figure 4: Rule Processing (78708 DL axioms)

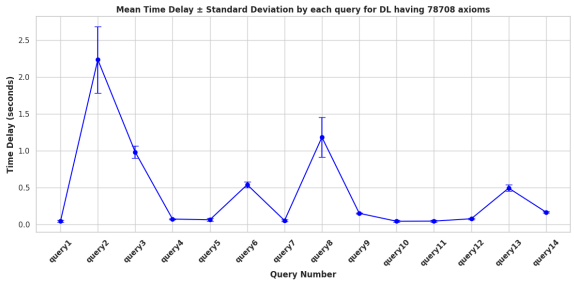


Figure 5: Average Query response time for 14 queries with a DL having 78708 axioms

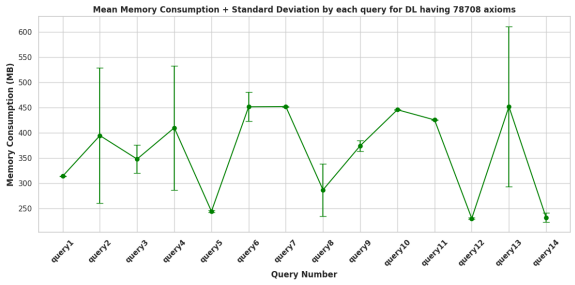


Figure 6: Average Memory Consumption for each of the 14 queries on a DL having 78,708 axioms.

Figure 7: Evaluation Result of NoHr Reasoner

Evaluation Summary: During the ontology preprocessing phase, a considerable amount of time is devoted to ontology classification and XSB loading, both of which are significantly affected by the number of DL axioms (figure 2). In the rule processing stage, the XSB loading time is directly proportional to the size of the translated ontology (figure 4). During preprocessing, we also noted increased memory usage and CPU utilization, both of which were correlated with the size of the knowledge base (figure 3). Once preprocessing is complete, query answering with XSB-Prolog becomes highly efficient, particularly for selective queries that filter results based on specific conditions and return only a small subset of answers (figure 5). However, we observed a noticeable decline in the performance of queries that required reasoning over recursive rules. Certain DL constructors, such as *Equivalence Relationships* and *Transitive Properties*, are translated into recursive rules, which significantly affect performance. Furthermore, elevated memory usage persisted in the query-answering phase (figure 6).

3.2 Usage

Excessive memory and CPU consumption by reasoners can pose significant challenges in aeronautics applications, where real-time performance and resource efficiency are critical. In this context, the use of the NoHr reasoner with larger knowledge bases becomes problematic due to the high memory demands observed during evaluation. NoHr operates by translating a Hybrid MKNF knowledge base into an equivalent logic program using a sequence of transformation algorithms. This logic program is then executed by the XSB-Prolog engine to perform query answering. To mitigate memory consumption, we propose using NoHr exclusively during a preprocessing phase to generate the transformed logic program. During system operation, only the resulting logic program is executed independently with XSB-Prolog. This approach significantly reduces runtime memory usage while maintaining reasoning capabilities, as the live system no longer depends on the full NoHr infrastructure. As shown in figure 1, the steps up to the file writing phase are performed externally to generate the translated file. Subsequently, this file is used directly within a Prolog environment during the live application to execute the use case.

An essential component in implementing the proposed strategy is managing the consistency-resolving mechanism, which utilizes knowledge base doubling and semi-negative transformations. Although this process is internally handled by the NoHr reasoner, which is capable of returning both consistent and contradictory results, our approach relies exclusively on the translated, doubled logic program executed independently within a Prolog engine. Therefore, a custom strategy must be developed within the Prolog environment to replicate the behavior originally facilitated by NoHr.

4 Additional Expressive Features

By analyzing various use cases, we identified the need for additional expressive features crucial for implementing aeronautics applications that are not supported in Hybrid MKNF by default. One such feature is the explicit representation of negative knowledge within the LP component of Hybrid MKNF. Consider the following case: "*A runway is operational only if there is no NOTAM indicating that it is closed, and no obstacle is present on the runway*". This can be initially represented by the following MKNF rule:

$$\mathbf{K}\ \mathrm{OpnRwy(X)} \leftarrow \mathbf{K}\ \mathrm{Rwy(X)},\ \mathbf{not}\ \mathrm{ob(A,X)},\ \mathbf{not}\ \mathrm{CldRwy(X)}. \tag{22}$$

However, due to the safety-critical nature of aeronautics, the predicate *ob(A, X)*, which indicates that there exists an obstacle on runway *X* in airport *A*, carries particular importance. Using **not** operator with *ob(A,X)* to infer that a runway is operational introduces safety concerns because it relies on the absence of evidence rather than explicit confirmation. Such assumptions are unacceptable in this domain. Instead,

we must explicitly verify that the runway is free of obstacles, thereby enhancing the reliability of the system. One way to achieve this is by replacing the **not** operator with classical negation denoted as $\neg$.

Classical negation can be incorporated into existing Hybrid MKNF semantics via syntactic transformation for unary and binary predicates. For a unary predicate A, we introduce a new concept B with the axiom $B \equiv \neg A$ in DL, allowing B to stand for the classical negation of A within rules. For a binary predicate P, a corresponding predicate Q is created, and the disjointness axiom $P \sqcap Q \sqsubseteq \bot$ enforces that Q represents the classical negation of P. However, this approach requires additional knowledge engineering and is limited to unary and binary predicates. So extending the rule part with a classical negation operator enhances expressiveness and also simplifies knowledge engineering and also allows for easier modeling of critical scenarios similar to the example provided above. After the extension of classical negation, the rule (22) can be rewritten as:

$$\mathbf{K}\ \text{OpnRwy(X)} \leftarrow \mathbf{K}\ \text{Rwy(X)}, \mathbf{K}\neg\text{ob(A,X)},\ \mathbf{not}\ \text{CldRwy(X)}. \tag{23}$$

Another crucial feature currently lacking is the capability to represent integrity constraints within the knowledge base and enforce them during query answering. In the following sections, we present heuristics for implementing classical negation and discuss a method to ensure that the query results comply with all specified integrity constraints within the context of Hybrid MKNF.

4.1 Classical Negation

Building on the insights from [35] and [2], along with existing contradictory resolution techniques, such as knowledge base doubling and semi-negative transformation, we propose a method to integrate classical negation into the LP component of a Hybrid MKNF knowledge base. In this context, the modified Hybrid MKNF is represented as $KB = (O,P)$, where O denotes the DL ontology and P comprises the rules of form (1). Here, H, A_m for $1 \leq i \leq m$ and B_n for $1 \leq j \leq n$ are first-order atoms that may be preceded by classical negation, as indicated by ($\neg$). In this revised version of the Hybrid MKNF, the LP rules now incorporate classically negative literals. As highlighted in [35], this classical negation can be eliminated through syntactic transformation, enabling the reuse of existing Hybrid MKNF algorithms for reasoning, which is the core concept of this proposal. The modified knowledge base KB' is obtained from KB by substituting each classically negated atom $\neg L$ in P with a new predicate L' and introducing constraints C in the form $\bot \leftarrow \mathbf{K}\,L, \mathbf{K}\,L'$. This results in a modified knowledge base, $KB' = (O, P' \cup C)$. Subsequently, knowledge base doubling and semi-negative transformation are applied, as detailed in [1], leading to the doubled knowledge base $KB'^d = (O, O^d, P'^d)$, as outlined below:

1. O^d is obtained by substituting each predicate L in O with L^d;
2. P^d is derived by transforming each rule of the form (1) occurring in P into two rules:
   i $\mathbf{K}\,H \leftarrow \mathbf{K}\,A_1, \ldots, \mathbf{K}\,A_i, \mathbf{not}\,B_1^d, \ldots, \mathbf{not}\,B_j^d$;
   ii $\mathbf{K}\,H^d \leftarrow \mathbf{K}\,A_1^d, \ldots, \mathbf{K}\,A_i^d, \mathbf{not}\,B_1, \ldots, \mathbf{not}\,B_j, \mathbf{not}\,NH$;
3. C^d is derived from each constraint of the form $\bot \leftarrow \mathbf{K}\,L, \mathbf{K}\,L'$ in C is transformed into two rules: $\mathbf{K}\,NL \leftarrow \mathbf{K}\,L'$ and $\mathbf{K}\,NL' \leftarrow \mathbf{K}\,L$. ($\bot \leftarrow \mathbf{K}\,L, \mathbf{K}\,L' \equiv \mathbf{K}\,\neg L \leftarrow \mathbf{K}\,L'$ and $\mathbf{K}\,\neg L' \leftarrow \mathbf{K}\,L$. Replace $\neg L$ and $\neg L'$ with NL and NL' .)

where A_i, A_i^d, B_j, B_j^d, H, and H^d may represent either L or L'.

Consider Hybrid MKNF knowledge base $KB_2 = (O,P)$, where O contains an assertion *Rwy(rw1)* and P includes rules (23) and a fact: $\mathbf{K}\,\neg$*ob(ifbo,rw1)*. According to the syntactic transformation, these rules can be converted into the following rules, resulting in the modified knowledge base $KB'_2 = (O, P' \cup C)$ as illustrated below (only the LP part is provided):

$$\mathbf{K}\ \mathrm{OpnRwy(X)} \leftarrow \mathbf{K}\ \mathrm{Rwy}(X), \mathbf{K}\ \mathrm{ob}'(A,X),\ \mathbf{not}\ \mathrm{CldRwy}(X). \tag{24}$$

$$\bot \leftarrow \mathbf{K}\ \mathrm{ob}(A,X), \mathbf{K}\ \mathrm{ob}'(A,X). \tag{25}$$

$$\mathbf{K}\ \mathrm{ob}'(lfbo, rw1). \tag{26}$$

After knowledge base doubling and semi-negative transformation we get $KB_2^{r^d} = (O, O^d, P^{r^d})$:

$$\mathbf{K}\ \mathrm{opnRwy(X)} \leftarrow \mathbf{K}\ \mathrm{rwy}(X), \mathbf{K}\ \mathrm{ob}'(A,X),\ \mathbf{not}\ \mathrm{cldRwy}^d(X). \tag{27}$$

$$\mathbf{K}\ \mathrm{opnRwy}^d(X) \leftarrow \mathbf{K}\ \mathrm{rwy}^d(X), \mathbf{K}\ \mathrm{ob}^{r^d}(A,X),\ \mathbf{not}\ \mathrm{cldRwy}(X), \mathbf{not}\ \mathrm{NopnRwy}(X). \tag{28}$$

$$\mathbf{K}\ \mathrm{Nob}(A,X) \leftarrow \mathbf{K}\ \mathrm{ob}'(A,X). \tag{29}$$

$$\mathbf{K}\ \mathrm{Nob}'(A,X) \leftarrow \mathbf{K}\ \mathrm{ob}(A,X). \tag{30}$$

$$\mathbf{K}\ \mathrm{ob}'(lfbo, rw1). \tag{31}$$

$$\mathbf{K}\ \mathrm{ob}^{r^d}(lfbo, rw1) \leftarrow \mathbf{not}\ \mathrm{Nob}'(lfbo, rw1). \tag{32}$$

After applying all transformations, we obtain the modified knowledge base $KB_2^{r^d} = (O, O^d, P^{r^d})$, where all occurrences of classical negation within the rules are syntactically eliminated. Despite the introduction of additional predicates and rules, we hypothesize that the proposed syntactic transformation preserves the semantic equivalence between the original Hybrid MKNF knowledge base with classical negation and the transformed negation-free knowledge base. This hypothesis is supported by existing approaches, such as [35] and [2], which follow the same transformation to address classical negation within a logic program, as well as the knowledge base doubling and semi-negative transformation proposed in [1] to handle classical negation within the DL part of Hybrid MKNF. Formal proof of semantic equivalence is the subject of ongoing work and is planned for future publications. Consequently, we can employ existing query answering algorithms such as those proposed in [26] and [7]. This is achieved by converting the DL part into its semantically equivalent LP rules P^O, then applying syntactic transformation to remove classical negation, $KB_2' = (\emptyset, P^O, P' \cup C)$, and subsequently apply knowledge base doubling and semi-negative transformation, and use the resulting program to perform query answering using XSB-Prolog.

By adding the assertion *rwy(rw1)* in KB_2, we can infer *opnRwy(rw1)* because there is an explicit negative fact that indicates the absence of an obstacle, specifically $\neg$*ob(lfbo,rw1)*, and no closure is indicated. Now, consider a scenario of contradiction, which is an additional assertion *ob(lfbo,rw1)* that also needs to be doubled. In this case, *ob(lfbo,rw1)* becomes contradictory because both the atom and its classical negation are true simultaneously. In such a situation, the rules defined in (29) and (30) ensure that both $Nob(X,Y)$ and $Nob'(X,Y)$ are true. Consequently, because of the semi-negative transformation, both $ob^{r^d}(lfbo, rw1)$ and $ob^d(lfbo, rw2)$ are inferred to be false. This allows the detection of atom *ob(lfbo,rw1)*, along with its duplicate *obr(lfbo,rw2)*, to be contradictory. Recall that an atom L is considered consistently true only if both L and its duplicate L^d are true. Then, the consistent query answering mechanisms described earlier can be used to correctly handle such situations and provide only consistent answers.

4.2 Integrity Constraints

Integrity constraints are conditions used to verify the validity of both explicit and inferred knowledge. They are not intended to produce new inferences, but rather to ensure data consistency [4]. Our objective is to explicitly represent these constraints and ensure that all atoms in the result of a given query comply with them. Any atom found to violate a constraint is identified as a constraint violation and is excluded from the final query response. Inspired by the work in [20] and [8], we propose representing integrity constraints as condition-action rules of the form:

$$A_1, \ldots, A_n,\ \mathrm{not}\ B_1, \ldots, \mathrm{not}\ B_m \Rightarrow a_1, \ldots, a_k \tag{33}$$

Here, $A_1,\ldots,A_n$ and not $B_1,\ldots,$not B_m represent the condition, while $a_1,\ldots,a_k$ denote the actions. Each action is represented by an atom $\mathbf{K}\,A$ or its classical negation $\mathbf{K}\,\neg A$. Consider a fact that is deemed consistent only if a specific condition is satisfied. When this condition is not satisfied, the atoms responsible for the violation are identified as conflicting with the defined constraints and are excluded from subsequent reasoning processes. This is achieved by representing the violated condition as a rule of the form (33), where the head contains the classical negation of the atoms to be marked as conflicting. When the body of the rule holds, it signifies a constraint violation, and the corresponding rule head is derived. As a result, both the original atoms and their classical negation may coexist in the knowledge base, leading to inconsistencies. The rule (33) is converted into a set of MKNF rules with **K** and **not** operator as below:

$$\begin{aligned}\mathbf{K}\,a_1 \leftarrow \mathbf{K}\,A_1,\ldots,\mathbf{K}\,A_n,\ \mathbf{not}\,B_1,\ldots,\mathbf{not}\,B_m\\ \ldots\\ \mathbf{K}\,a_k \leftarrow \mathbf{K}\,A_1,\ldots,\mathbf{K}\,A_n,\ \mathbf{not}\,B_1,\ldots,\mathbf{not}\,B_m\end{aligned}$$

After the conversion, the constraint is treated as a standard MKNF rule. To handle classical negation, we apply the proposed syntactic transformation techniques and incorporate an inconsistency resolution strategy based on knowledge base doubling and semi-negative transformation. When the body of the translated MKNF rule evaluates to true, the head infers the negation of an atom, indicating a violation. This leads to contradictions in the MKNF knowledge base, where both an atom and its negation are simultaneously present. Consequently, by applying the consistency resolution strategy only consistent answers are produced during query answering, thereby removing all contradictory answers. This feature has several applications in the aeronautics domain. One such application involves identifying incomplete NOTAMs, such as those missing critical information like the start time, end time, or issuing agency. These incomplete messages are considered inconsistent and are excluded from the reasoning process to preserve the integrity of decisions made by the system. An example of an integrity constraint that enforces a basic validity condition on NOTAM messages is: "*A NOTAM must include both a start time and an end time*".

$$\mathbf{K}\,\neg\mathsf{Notam}(X) \leftarrow \mathbf{K}\,\mathsf{Notam}(X),\ \mathbf{not}\,\mathsf{hasStartTime}(X,Y),\ \mathbf{not}\,\mathsf{hasEndTime}(X,Z). \tag{34}$$

According to rule (34), if either the start time or end time is absent, the NOTAM is deemed inconsistent and is consequently exclude them from the further reasoning process. Another application is the detection of contradictory NOTAMs, for example, when two NOTAMs issued by different agencies refer to the same runway during the same time period, but one indicates that the runway is closed while the other states that it is operational. Integrity constraints with repair action will help to identify such NOTAMs and excluded from further reasoning process.

5 System Description

Figure 8 illustrates the simplified architecture of the system. We developed a preprocessing tool that utilizes the translator embedded in the NoHr reasoner to transform DL ontology into an equivalent set of rules. This process involves knowledge base doubling and semi-negative transformation. During translation, the NoHr translator systematically mapped each concept and role from the DL ontology into two distinct predicates representing both the original and doubled versions. For example, *Rwy* is mapped to *aRwy* to represent the original predicate, and to *dRwy* to represent the doubled predicate. Furthermore, all predicates in the translated program are declared tabled, and the standard **not** operator is replaced

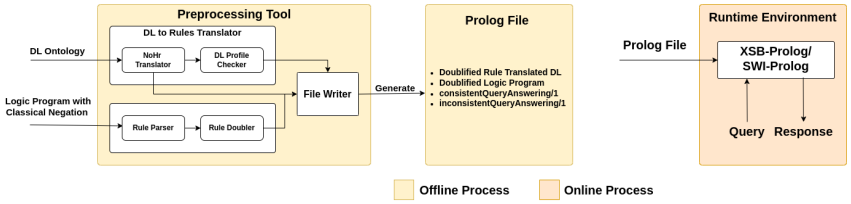


Figure 8: Architecture of Proposed Work Flow

with tabled negation, denoted by **tnot**. For example, DL axiom (4) is translated into rules (11) and (12), which appear in the Prolog program below, as generated by the NoHr translator.

```
1 arwy(X):- aopnRwy(X).
2 drwy(X):- dopnRwy(X), tnot(nrwy(X)).
```

Classical negation in rules can be represented using the symbol "-". An additional parser is required to handle this classical negation in the LP part of Hybrid MKNF. This parser eliminates classical negation according to the proposed syntactic transformation technique, and subsequently applies both knowledge base doubling and semi-negative transformation. Furthermore, it designates all atoms as tabled, and alters the **not** operator to **tnot**. The rule parser converts rules (27) and (28) into a prolog program as follows (see Section 6.1): $\neg ob(X)$ is substituted with a newly introduced predicate *nonob(X)* by adding a prefix "*non*" instead of $\neg$.

```
1 aopnRwy(X):- arwy(X),anonob(A,X),tnot(dcldRwy(X)).
2 dopnRwy(X):- drwy(X),dnonob(A,X),tnot(acldRwy(X)),tnot(nopnRwy(X)).
```

To implement query answering, two meta-predicates are introduced: *consistentQuery/1*, which returns consistent answers, and *inconsistentQuery/1*, which returns contradictory answers. These meta-predicates implement query doubling, as explained in (20) and (21), by matching them with transformed logic programs. After doubling, these meta-predicates implement the conditions $q(X) \wedge q^d(X)$ and $q(X) \wedge$ **not** $q^d(X)$ to query the consistent and contradictory answers, respectively. The built-in Prolog predicate *atom_concat/3* is employed to prepend additional strings, which is beneficial for query doubling. Subsequently, the *call/1* predicate is used to execute the transformed queries. The *file writer module* integrates translated DL with predefined meta-predicates and doubled logic program from parser, exports to a file executable by Prolog engines. Also it generates a report on unsupported axioms in the DL ontology, with examples available at `https://drive.uca.fr/d/040c6ecbe940449e9c0d/`. After translation, the doubled LP part along with the translated DL axioms can be executed using a Prolog engine that supports well-founded semantics. For query answering, we used SWI-Prolog or XSB-Prolog for evaluation. figures 9 and 10 illustrate the query response time and memory consumption during the evaluation of each query for XSB-Prolog and SWI-Prolog, respectively. Overall, we observe a substantial reduction in memory usage across all queries, except for the first execution of more complex recursive queries. This is because the tabling feature stores previous results after the initial computation, allowing subsequent executions of the same query to reuse these results without repeating the full computation. Preprocessed files and queries used for this evaluation are available here: `https://drive.uca.fr/d/e5482dd1af5f4fdd8bfb/`.

6 Related Work

In general, three main approaches exist for combining DLs with LP: loose integration, tight integration, and full integration [14]. Each of these integration strategies can be based on one of two major seman-

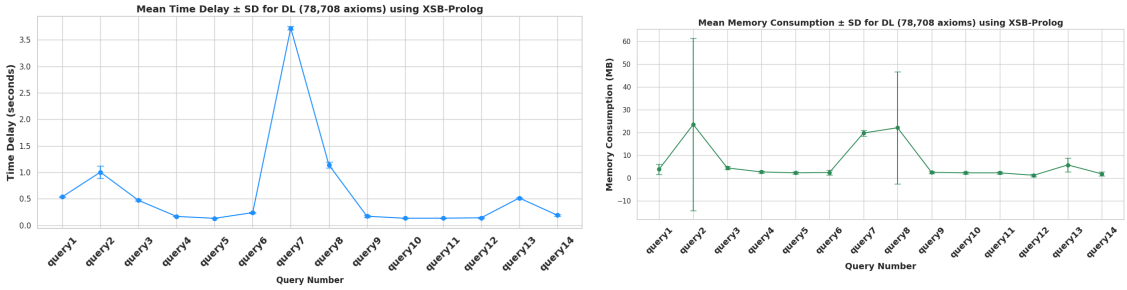


Figure 9: Response Time and Memory Consumption for each query with XSB-Prolog

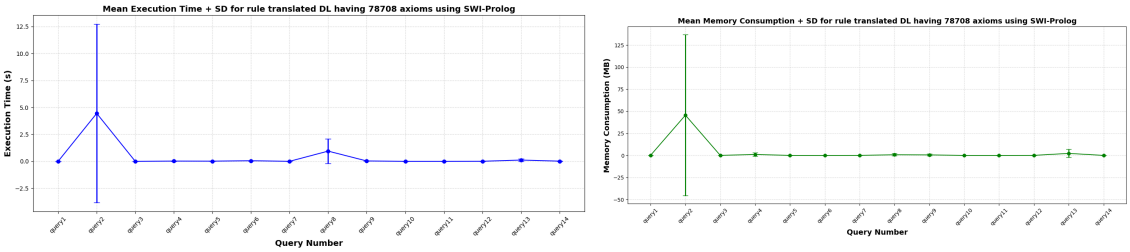


Figure 10: Response Time and Memory Consumption for each query with SWI-Prolog

tics: well-founded [21] or answer set semantics [12]. In loose integration, a DL ontology and LP rules are regarded as two separate entities, with a specified interface facilitating the exchange of knowledge. DL-Programs is a loose integration approach that proposed under both answer set semantics [16] and well-founded semantics [15]. In DL-Programs, knowledge exchange between DL and LP rules is facilitated only through external operators, necessitating additional rules that increase the language's complexity for knowledge engineers. The only reasoning task in DL-Programs is model computation, where the resulting model is stored in a database to enable query answering. However, this approach is not memory-efficient, as it requires computing and storing the full model. Nonetheless, DL-Programs offer the benefit of integrating with external data sources, such as Python programs or relational databases. An extended language known as *HEX* [18], along with its corresponding reasoner, *DLV-Hex* [17], supports this integration. There also exists an optimized reasoner called Drew System [44] that utilizes two datalog rewritable DLs, $\mathscr{LDL}+$ [22] and $\mathscr{SROEL}$ [31] for DL-Programs. Additionally, an alternative loose integration approach called F-Logic# has been proposed that combines F-logic with DL ontologies [23].

In tight integration, an integrated model that satisfies both the DL and LP parts is defined. The integrated model is $M = M_O \cup M_L$, where M_O satisfies the DL ontology and M_L satisfies the corresponding LP rules. We start with a model of the DL part, denoted as M_O. Using this model, we simplify the grounded LP part in two steps. First, we remove any LP rules that contain DL atoms in their body that are false in M_O, since those rules cannot be satisfied. Then, from the remaining rules, we remove any DL atoms in the body that are true in M_O, as they are already satisfied. This results in a version of the LP program that no longer includes any DL atoms. We then compute a model for this simplified LP using either answer set or well-founded semantics. This process is repeated for each possible DL model M_O, producing a corresponding LP model for each one. Several hybrid languages have been developed with a focus on tight integration. Notably, R-Hybrid KB [40] and DL+Log [41] are founded on answer set semantics, while HD-Rule is based on well-founded semantics [11]. Furthermore, an

extension of DL+Log has been introduced, which incorporates closed-world predicates within a DL ontology, leading to the creation of clopen knowledge bases [6]. Additionally, Resilient Logic Programs (RLP) have been proposed to merge both tight and loose integration, where loose integration shares only logical consequences between DL and LP, while tight integration enables knowledge sharing between individual models [36]. In tight integration, even when employing well-founded semantics, numerous possible models may exist, complicating the reasoning process. Consequently, constructing a practical reasoner under these conditions is challenging. Additionally, the knowledge engineer must possess an understanding of individual models of DLs to generate the corresponding rule program due to the model-based integration between DL and LP, further complicating the knowledge engineering process.

In full integration, a singular unifying non-monotonic formalism is defined to encapsulate both the semantics of DLs and LP. Several non-monotonic extensions of first-order logic have been proposed, including default logic, epistemic logic, defeasible logic, and circumscription. Ideally, selecting one of these extensions to integrate both DL and rules constitutes the fundamental concept of this approach. In full integration, we encounter works grounded in Minimal Knowledge and Negation-as-a-failure (MKNF), Open-Answer Set Programming (OASP) [24], and Defeasible Logics [3]. We intentionally excluded a significant array of research that merges DLs with first-order rules, such as SWRL [32], AL-Log [9], Carin [33], due to their lack of closed-world reasoning capabilities. Given the undecidability of OASP, several decidable variants have been proposed. These include the F-Hybrid Knowledge Base, which integrates $\mathscr{SHOQ}$ DL with a finite set of Forest Logic Programs (FLP) [19]. These programs allow only unary and binary predicates and adhere to the forest-model property. Additionally, the G-Hybrid knowledge base combines $\mathscr{DLROI}^{-\{\leq\}}$ with a finite set of ASP rules that comply with the guardness restriction, wherein each rule contains a guarded predicate, and all variables within the rules must appear in that predicate [25]. In this context, Hybrid MKNF offers a more general and expressive language that integrates DLs and LP [38].

7 Conclusion

We introduced Hybrid MKNF as a knowledge representation language tailored for aeronautics applications, such as NARS. To address memory constraints in aeronautic systems, we propose an alternative workflow that employs the NoHr reasoner through offline preprocessing. Our approach manages both consistent and contradictory query responses by using Prolog meta-predicates. We developed heuristics to enhance Hybrid MKNF with classical negation and to support integrity constraints through repair actions. A tool was created to convert a Hybrid MKNF knowledge base into an equivalent set of LP rules using NoHr by integrating these heuristics. In future research, we plan to formally define the integration of classical negation in the LP into Hybrid MKNF semantics under well-founded semantics. This study aims to identify syntactic fragments of Hybrid MKNF, such as definite and stratified programs, that are compatible with lightweight rule engines like I-DLV, Trealla Prolog, and Tau-Prolog. The goal is to enable efficient reasoning on resource-constrained platforms and to guide the selection of suitable rule engines based on expressivity requirements. We will enhance explainability by adapting provenance-based techniques to Hybrid MKNF under well-founded semantics to provide reasoning justifications. Finally, we will incorporate temporal reasoning into Hybrid MKNF for scenarios such as NOTAMs, where facts are valid over specific time intervals, by extending the framework with temporal operators.

References

[1] José Júlio Alferes, Matthias Knorr & Terrance Swift (2013): *Query-Driven Procedures for Hybrid MKNF Knowledge Bases*. *ACM Trans. Comput. Log.* 14(2), pp. 16:1–16:43, doi:10.1145/2480759.2480768.

[2] José Júlio Alferes, Luís Moniz Pereira & Teodor C. Przymusinski (1998): *'Classical' Negation in Nonmonotonic Reasoning and Logic Programming*. *J. Autom. Reason.* 20(1), pp. 107–142, doi:10.1023/A:1005900924623.

[3] Grigoris Antoniou & Antonis Bikakis (2007): *DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the Semantic Web*. *IEEE Trans. Knowl. Data Eng.* 19(2), pp. 233–245, doi:10.1109/TKDE.2007.29.

[4] Patrizia Asirelli, Michelle De Santis & Maurizio Martelli (1985): *Integrity constraints in logic databases*. *The Journal of Logic Programming* 2(3), pp. 221–232, doi:10.1016/0743-1066(85)90020-2. Available at `https://www.sciencedirect.com/science/article/pii/0743106685900202`.

[5] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi & Peter F. Patel-Schneider, editors (2003): *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press.

[6] Labinot Bajraktari, Magdalena Ortiz & Mantas Simkus (2018): *Combining Rules and Ontologies into Clopen Knowledge Bases*. In Sheila A. McIlraith & Kilian Q. Weinberger, editors: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, AAAI Press, pp. 1728–1735, doi:10.1609/aaai.v32i1.11565.

[7] Nuno Costa, Matthias Knorr & João Leite (2015): *Next Step for NoHR: OWL 2 QL*. In Marcelo Arenas, Óscar Corcho, Elena Simperl, Markus Strohmaier, Mathieu d'Aquin, Kavitha Srinivas, Paul Groth, Michel Dumontier, Jeff Heflin, Krishnaprasad Thirunarayan & Steffen Staab, editors: *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, *Lecture Notes in Computer Science* 9366, Springer, pp. 569–586, doi:10.1007/978-3-319-25007-6_33.

[8] Luís Cruz-Filipe, Michael Franz, Artavazd Hakhverdyan, Marta Ludovico, Isabel Nunes & Peter Schneider-Kamp (2015): *Active Integrity Constraints: From Theory to Implementation*. In Ana L. N. Fred, Jan L. G. Dietz, David Aveiro, Kecheng Liu & Joaquim Filipe, editors: *Knowledge Discovery, Knowledge Engineering and Knowledge Management - 7th International Joint Conference, IC3K 2015, Lisbon, Portugal, November 12-14, 2015, Revised Selected Papers*, *Communications in Computer and Information Science* 631, Springer, pp. 399–420, doi:10.1007/978-3-319-52758-1_22.

[9] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi & Andrea Schaerf (1998): *AL-log: Integrating Datalog and Description Logics*. *J. Intell. Inf. Syst.* 10(3), pp. 227–252, doi:10.1023/A:1008687430626.

[10] Wlodzimierz Drabent (2010): *Hybrid Reasoning with Non-monotonic Rules*. In Uwe Aßmann, Andreas Bartho & Christian Wende, editors: *Reasoning Web. Semantic Technologies for Software Engineering, 6th International Summer School 2010, Dresden, Germany, August 30 - September 3, 2010. Tutorial Lectures*, *Lecture Notes in Computer Science* 6325, Springer, pp. 28–61, doi:10.1007/978-3-642-15543-7_2.

[11] Wlodzimierz Drabent, Jakob Henriksson & Jan Maluszynski (2007): *HD-rules: A Hybrid System Interfacing Prolog with DL-reasoners*. In Axel Polleres, David Pearce, Stijn Heymans & Edna Ruckhaus, editors: *Proceedings of the ICLP'07 Workshop on Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services, ALPSWS 2007, Porto, Portugal, September 13th, 2007*, *CEUR Workshop Proceedings* 287, CEUR-WS.org. Available at `https://ceur-ws.org/Vol-287/paper_9.pdf`.

[12] Thomas Eiter, Giovambattista Ianni & Thomas Krennwallner (2009): *Answer Set Programming: A Primer*. In Sergio Tessaris, Enrico Franconi, Thomas Eiter, Claudio Gutierrez, Siegfried Handschuh, Marie-Christine Rousset & Renate A. Schmidt, editors: *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial*

Lectures, *Lecture Notes in Computer Science* 5689, Springer, pp. 40–110, doi:10.1007/978-3-642-03754-2_-2.

[13] Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner & Axel Polleres (2008): *Rules and Ontologies for the Semantic Web*. In Cristina Baroglio, Piero A. Bonatti, Jan Maluszynski, Massimo Marchiori, Axel Polleres & Sebastian Schaffert, editors: *Reasoning Web, 4th International Summer School 2008, Venice, Italy, September 7-11, 2008, Tutorial Lectures*, *Lecture Notes in Computer Science* 5224, Springer, pp. 1–53, doi:10.1007/978-3-540-85658-0_1.

[14] Thomas Eiter, Giovambattista Ianni, Thomas Krennwallner & Axel Polleres (2008): *Rules and Ontologies for the Semantic Web*. In Cristina Baroglio, Piero A. Bonatti, Jan Maluszynski, Massimo Marchiori, Axel Polleres & Sebastian Schaffert, editors: *Reasoning Web, 4th International Summer School 2008, Venice, Italy, September 7-11, 2008, Tutorial Lectures*, *Lecture Notes in Computer Science* 5224, Springer, pp. 1–53, doi:10.1007/978-3-540-85658-0_1.

[15] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz & Roman Schindlauer (2011): *Well-founded semantics for description logic programs in the semantic web*. *ACM Trans. Comput. Log.* 12(2), pp. 11:1–11:41, doi:10.1145/1877714.1877717.

[16] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer & Hans Tompits (2008): *Combining answer set programming with description logics for the Semantic Web*. *Artificial Intelligence* 172(12), pp. 1495–1539, doi:10.1016/j.artint.2008.04.002.

[17] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer & Hans Tompits (2005): *DLV-HEX : Dealing with Semantic Web under Answer-Set Programming*. Available at `https://api.semanticscholar.org/CorpusID:9592774`.

[18] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer & Hans Tompits (2005): *A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer-Set Programming*. In Leslie Pack Kaelbling & Alessandro Saffiotti, editors: *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, Professional Book Center, pp. 90–96. Available at `http://ijcai.org/Proceedings/05/Papers/1353.pdf`.

[19] Cristina Feier & Stijn Heymans (2013): *Reasoning with Forest Logic Programs and f-hybrid knowledge bases*. *Theory Pract. Log. Program.* 13(3), pp. 395–463, doi:10.1017/S1471068411000597.

[20] Sergio Flesca, Sergio Greco & Ester Zumpano (2004): *Active integrity constraints*. In: *Proceedings of the 6th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, PPDP '04, Association for Computing Machinery, New York, NY, USA, p. 98–107, doi:10.1145/1013963.1013977.

[21] Allen Van Gelder, Kenneth A. Ross & John S. Schlipf (1991): *The Well-Founded Semantics for General Logic Programs*. *J. ACM* 38(3), pp. 620–650, doi:10.1145/116825.116838.

[22] Stijn Heymans, Thomas Eiter & Guohui Xiao (2010): *Tractable Reasoning with DL-Programs over Datalog-rewritable Description Logics*. In Helder Coelho, Rudi Studer & Michael J. Wooldridge, editors: *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, *Frontiers in Artificial Intelligence and Applications* 215, IOS Press, pp. 35–40. Available at `http://www.booksonline.iospress.nl/Content/View.aspx?piid=17710`.

[23] Stijn Heymans, Roman Korf, Michael Erdmann, Jörg Pührer & Thomas Eiter (2010): *F-Logic#: Loosely Coupling F-Logic Rules and Ontologies*. In Jimmy X. Huang, Irwin King, Vijay V. Raghavan & Stefan M. Rüger, editors: *WI 2010 - 2010 IEEE/WIC/ACM International Conference on Web Intelligence, Toronto, Canada, August 31 - September 3, 2010, Main Conference Proceedings*, IEEE Computer Society, pp. 248–255, doi:10.1109/WI-IAT.2010.44.

[24] Stijn Heymans, Davy Van Nieuwenborgh & Dirk Vermeir (2007): *Open answer set programming for the semantic web*. *J. Appl. Log.* 5(1), pp. 144–169, doi:10.1016/J.JAL.2006.02.001.

[25] Stijn Heymans, Livia Predoiu, Cristina Feier, Jos de Bruijn & Davy Van Nieuwenborgh (2003): *G-Hybrid Knowledge Bases*. Available at `https://api.semanticscholar.org/CorpusID:9440798`.

[26] Vadim Ivanov, Matthias Knorr & João Leite (2013): *A Query Tool for* EL *with Non-monotonic Rules*. In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha F. Noy, Chris Welty & Krzysztof Janowicz, editors: *The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I*, *Lecture Notes in Computer Science* 8218, Springer, pp. 216–231, doi:10.1007/978-3-642-41335-3_14.

[27] Vedran Kasalica, Matthias Knorr, João Leite & Carlos Lopes (2020): *NoHR: An Overview*. *Künstliche Intell.* 34(4), pp. 509–515, doi:10.1007/S13218-020-00650-1.

[28] Matthias Knorr (2021): *On Combining Ontologies and Rules*. In Mantas Simkus & Ivan Varzinczak, editors: *Reasoning Web. Declarative Artificial Intelligence - 17th International Summer School 2021, Leuven, Belgium, September 8-15, 2021, Tutorial Lectures*, *Lecture Notes in Computer Science* 13100, Springer, pp. 22–58, doi:10.1007/978-3-030-95481-9_2.

[29] Matthias Knorr, José Júlio Alferes & Pascal Hitzler (2008): *A Coherent Well-founded Model for Hybrid MKNF Knowledge Bases*. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis & Nikolaos M. Avouris, editors: *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*, *Frontiers in Artificial Intelligence and Applications* 178, IOS Press, pp. 99–103, doi:10.3233/978-1-58603-891-5-99.

[30] Matthias Knorr, José Júlio Alferes & Pascal Hitzler (2011): *Local closed world reasoning with description logics under the well-founded semantics*. *Artif. Intell.* 175(9-10), pp. 1528–1554, doi:10.1016/j.artint.2011.01.007.

[31] Markus Krötzsch (2011): *Efficient Rule-Based Inferencing for OWL EL*. In Toby Walsh, editor: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, IJCAI/AAAI, pp. 2668–2673, doi:10.5591/978-1-57735-516-8/IJCAI11-444.

[32] Hong Lan (, 2004): *SWRL : A semantic Web rule language combining OWL and ruleML*. Available at `https://api.semanticscholar.org/CorpusID:207974723`.

[33] Alon Y. Levy & Marie-Christine Rousset (1998): *Combining Horn Rules and Description Logics in CARIN*. *Artif. Intell.* 104(1-2), pp. 165–209, doi:10.1016/S0004-3702(98)00048-4.

[34] Vladimir Lifschitz (1991): *Nonmonotonic Databases and Epistemic Queries*. In John Mylopoulos & Raymond Reiter, editors: *Proceedings of the 12th International Joint Conference on Artificial Intelligence. Sydney, Australia, August 24-30, 1991*, Morgan Kaufmann, pp. 381–386. Available at `http://ijcai.org/Proceedings/91-1/Papers/059.pdf`.

[35] Vladimir Lifschitz, David Pearce & Agustín Valverde (2001): *Strongly equivalent logic programs*. *ACM Trans. Comput. Log.* 2(4), pp. 526–541, doi:10.1145/383779.383783.

[36] Sanja Lukumbuzya, Magdalena Ortiz & Mantas Simkus (2020): *Resilient Logic Programs: Answer Set Programs Challenged by Ontologies*. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, AAAI Press, pp. 2917–2924, doi:10.1609/aaai.v34i03.5683.

[37] Boris Motik, Ian Horrocks, Riccardo Rosati & Ulrike Sattler (2006): *Can OWL and Logic Programming Live Together Happily Ever After?* In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold & Lora Aroyo, editors: *The Semantic Web - ISWC 2006, 5th International Semantic Web Conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006, Proceedings*, *Lecture Notes in Computer Science* 4273, Springer, pp. 501–514, doi:10.1007/11926078_36.

[38] Boris Motik & Riccardo Rosati (2007): *A Faithful Integration of Description Logics with Logic Programming*. In Manuela M. Veloso, editor: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pp. 477–482. Available at `http://ijcai.org/Proceedings/07/Papers/075.pdf`.

[39] Boris Motik & Riccardo Rosati (2010): *Reconciling description logics and rules*. *J. ACM* 57(5), pp. 30:1–30:62, doi:10.1145/1754399.1754403.

[40] Riccardo Rosati (2005): *On the decidability and complexity of integrating ontologies and rules*. *J. Web Semant.* 3(1), pp. 61–73, doi:10.1016/j.websem.2005.05.002.

[41] Riccardo Rosati (2006): *DL+log: Tight Integration of Description Logics and Disjunctive Datalog*. In Patrick Doherty, John Mylopoulos & Christopher A. Welty, editors: *Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006*, AAAI Press, pp. 68–78. Available at `http://www.aaai.org/Library/KR/2006/kr06-010.php`.

[42] Jessika Schulze (2016): *Handbook Of Logic In Artificial Intelligence And Logic Programming*. Available at `https://api.semanticscholar.org/CorpusID:63249517`.

[43] Gunjan Singh, Sumit Bhatia & Raghava Mutharaju (2020): *OWL2Bench: A Benchmark for OWL 2 Reasoners*. In Jeff Z. Pan, Valentina Tamma, Claudia d'Amato, Krzysztof Janowicz, Bo Fu, Axel Polleres, Oshani Seneviratne & Lalana Kagal, editors: *The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part II*, *Lecture Notes in Computer Science* 12507, Springer, pp. 81–96, doi:10.1007/978-3-030-62466-8_6.

[44] Guohui Xiao, Thomas Eiter & Stijn Heymans (2012): *The DReW System for Nonmonotonic DL-Programs*. In Juanzi Li, Guilin Qi, Dongyan Zhao, Wolfgang Nejdl & Hai-Tao Zheng, editors: *Semantic Web and Web Science - 6th Chinese Semantic Web Symposium and 1st Chinese Web Science Conference, CSWS 2012, Shenzhen, China, November 28-30, 2012*, Springer, pp. 383–390, doi:10.1007/978-1-4614-6880-6_33.

An ASP-based Solution to the Medical Appointment Scheduling Problem

Alina Vozna
University of Pisa, Italy
Department of Information Engineering,
Computer Science and Mathematics, University of L'Aquila,Italy
alina.vozna@student.univaq.it

Andrea Monaldini
University of Pisa, Italy
Department of Information Engineering,
Computer Science and Mathematics, University of L'Aquila,Italy
andrea.monaldini@student.univaq.it

Stefania Costantini
Department of Information Engineering,
Computer Science and Mathematics, University of L'Aquila,Italy
stefania.costantini@univaq.it

Valentina Pitoni
Department of Information Engineering,
Computer Science and Mathematics, University of L'Aquila,Italy
valentina.pitoni@univaq.it

Dawid Pado
Department of Information Engineering,
Computer Science and Mathematics, University of L'Aquila,Italy
dawid.pado@student.univaq.it

This paper presents an Answer Set Programming (ASP)-based framework for medical appointment scheduling, aimed at improving efficiency, reducing administrative overhead, and enhancing patient-centered care. The framework personalizes scheduling for vulnerable populations by integrating Blueprint Personas. It ensures real-time availability updates, conflict-free assignments, and seamless interoperability with existing healthcare platforms by centralizing planning operations within an ASP logic model.

1 Introduction

In modern healthcare systems, efficient management of medical appointments is a critical challenge to ensure timely and equitable access to care. The increasing complexity of healthcare delivery, driven by aging populations, the increasing prevalence of chronic diseases, and limited medical resources, has underscored the limitations of traditional scheduling methods. These systems often fail to consider the multitude of clinical, logistic and personal constraints involved in appointment assignment. Key goals such as reducing wait times, improving patient satisfaction and optimizing resource utilization are frequently compromised by static first-available allocation rules and lack of personalization.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 367–382, doi:10.4204/EPTCS.439.25

© Vozna, Monaldini, Costantini, Pitoni & Pado
This work is licensed under the
Creative Commons Attribution License.

Conventional systems such as Italy's centralized reservation infrastructure (Centro Unico di Prenotazione – CUP) exemplify these shortcomings. Appointments are typically assigned without considering medical urgency, patient location, or individual preferences. This results in inefficient scheduling, high no-show rates, and increased administrative overhead for rescheduling and manual adjustments. Additionally, rigid frameworks lack the flexibility to handle real-time variations such as staff shortages or emergency priorities, leading to delayed care and resource misallocation.

To address these challenges, this paper proposes an automated, constraint-based scheduling system using Answer Set Programming (ASP) [6, 20, 9, 21]. ASP is a declarative logic programming paradigm well-suited for solving NP-complete problems involving complex, dynamic constraints. Using the expressive power of ASP, the proposed system automatically eliminates infeasible solutions, such as inaccessible facilities for disabled patients or incompatible temporally slots, and generates optimal schedules that minimize penalties associated with distance, waiting time, sensory discomfort, and preference violations.

A distinctive feature of this work is the integration of Blueprint Personas [23], structured models that capture the clinical, social, and behavioral attributes of different patient archetypes. Originally from European digital health initiatives, Blueprint Personas support patient-centered care by incorporating real-world constraints such as cognitive limitations, accessibility needs, and socioeconomic conditions into the scheduling process. Their use improves equity, personalization, and adherence to care plans, particularly for vulnerable populations.

The ASP-based system developed here is designed for deployment in centralized or semi-centralized healthcare infrastructures, making it particularly applicable to national systems like the CUP. Although CUP facilitates centralized access, it assigns appointments using simplistic logic, often not accommodating crucial factors such as urgency, patient preferences, or accessibility constraints [14]. In contrast, our approach supports real-time rescheduling, emergency prioritization, and ethical decision making capabilities essential in modern healthcare environments, especially under conditions of resource scarcity or high demand [4]. Even though the system is well suited to centralized infrastructures like Italy's CUP, it can also be adapted to federated systems by decentralizing the scheduling logic across multiple institutions, maintaining consistency through shared constraints.

Furthermore, this work demonstrates how ASP can support explainable, multi-objective optimization in appointment scheduling. Beyond traditional rule-based or heuristic methods, ASP provides full declarative modeling, enabling systematic integration of patient preferences, medical priorities, and organizational constraints within a scalable and adaptable framework. The solution is modular and generalizable to other domains such as workforce management and educational timetabling.

The paper is structured as follows: Section 2 discusses background and related work in the scheduling of medical appointments and ASP-based optimization; Section 3 presents the formulation of the problem, detailing the constraints and objectives of the scheduling model; Section 4 describes the ASP-based system architecture and implementation; Section 5 provides an empirical evaluation of the approach, comparing its performance with conventional scheduling methods; finally, in Section 6 we conclude and discuss future research directions.

2 Related work

Appointment scheduling has been extensively studied through queueing theory, integer programming, and simulation models, aiming to address common challenges such as service time variability, patient no-shows, and limited healthcare resources [14]. More recent works have introduced decision-support tools

to optimize scheduling in outpatient contexts [25], integrating data-driven and logic-based techniques to manage complex constraints.

Various optimization approaches, such as Mixed-Integer Linear Programming (MILP), Answer Set Programming (ASP), and metaheuristic algorithms—have shown promising results in handling overbooking strategies, resource balancing, and patient flow management. Stochastic programming, in particular, has been used to optimize patient access while maintaining efficient resource use [19]. Heuristic techniques like Fuzzy Ant Lion Optimization (FALO) have improved fairness and reduced patient waiting times without compromising resource efficiency [3]. Discrete Event Simulation (DES), combined with patient behavior modeling and genetic algorithms, has also enhanced scheduling adaptability to preferences, cancellations, and dynamic demand [12].

ASP has proven particularly effective in resource constrained healthcare scenarios, including operating room scheduling [10] and nurse rostering [5]. Notably, Cappanera et al. [8] leveraged Logic-Based Benders Decomposition (LBBD) within ASP to manage the scheduling of chronic outpatients with comorbidities, improving scalability by decoupling day-level planning from daily agenda construction. Similarly, Kanias [17] developed a fairness-aware ASP-based system that integrates SQL databases via Python-CLORM and supports large-scale medical appointment rescheduling, focusing on maximizing community benefit.

Artificial intelligence (AI) methods have further advanced scheduling capabilities. Reinforcement learning and deep learning have been applied to learn optimal policies from historical appointment data, enabling dynamic adaptation to evolving patient behavior. In particular, Ala et al. [2] propose a deep learning framework based on Bi-LSTM (Bidirectional Long Short-Term Memory) and reinforcement learning to optimize appointment scheduling under uncertainty. The Bi-LSTM network is used to predict patient behavior, such as likelihood of attendance or preferred time windows, by analyzing historical appointment logs. These predictions are then used within a reinforcement learning loop to dynamically update scheduling policies. The model aims to minimize waiting times and reduce idle resources in complex clinical systems.

While these learning-based methods are powerful for uncovering temporal patterns, they often require large volumes of structured training data and lack interpretability in decision logic. Moreover, they do not natively support rich symbolic constraints such as accessibility needs, fairness rules, or ethical priorities. Our ASP-based approach complements these limitations by enabling transparent, rule-based reasoning over patient profiles, preferences, and clinical policies—allowing better explainability and integration with constraint-heavy scenarios.

Hybrid AI models—such as those combining evolutionary algorithms with two-stage stochastic programming—have been used in high-uncertainty environments like MRI scheduling [27]. Additionally, decentralized approaches using Distributed Constraint Optimization Problems (DCOPs) have framed scheduling as a multi-agent negotiation process [16].

However, despite the sophistication of these techniques, a persistent gap remains between theoretical models and real-world healthcare operations. Many frameworks overlook critical practical factors such as fluctuating clinician availability, patient heterogeneity, and the administrative burden of rescheduling [18]. A significant differentiator of our work lies in its alignment with real-world healthcare infrastructure and policy, particularly through the lens of national systems. For example, in Italy, appointment scheduling is largely managed through the Centro Unico di Prenotazione (CUP), a centralized system that assigns the first available appointment without considering urgency, accessibility, or patient preferences. This results in rigid workflows, high no-show rates, and inefficiencies during unforeseen events such as clinician absences or late cancellations. In contrast, several international systems incorporate varying degrees of flexibility and personalization. The UK's e-RS platform, for instance, allows patients

to select their preferred date, time, and location; France's Doctolib offers digital access to video consultations and personalized appointment settings; and Germany's eTerminservice ensures prioritization for urgent referrals. These systems, although more adaptive, often rely on custom integrations and lack the logical rigor and constraint expressivity that ASP offers.

Unlike heuristic, machine learning, or stochastic approaches, our ASP-based scheduling framework guarantees full compliance with hard constraints, provides explainable decision-making, and supports real-time adaptability to changing healthcare needs. Moreover, while many prior systems focus on optimizing throughput or resource utilization, few holistically account for patient-centered constraints—such as sensory preferences, accessibility requirements, multi-session planning, and financial limitations. Our solution bridges this gap by modeling scheduling as a declarative logic problem, ensuring ethical and personalized healthcare access within realistic operational boundaries.

3 Problem Description and Formalization

Effectively managing medical appointments is a complex, multidimensional problem that requires balancing clinical urgency, limited medical resources, patient-specific preferences, and logistical constraints. Traditional scheduling approaches often fall short, as they struggle to reconcile these competing factors within a unified, adaptable framework.

To overcome these limitations, we formulate appointment scheduling as a multi-objective constrained optimization problem and solve it using Answer Set Programming (ASP). The declarative nature of ASP allows for an elegant and compact encoding of domain knowledge, enabling automated reasoning over a vast space of possible solutions while strictly enforcing all constraints.

The problem centers around three core entities: clinics, which offer appointment slots under fixed budget limits; doctors, whose availability and workloads must be balanced; and patients, who vary in urgency (high, medium, low) and may express preferences regarding time windows, clinic locations, specific physicians, or even environmental conditions (e.g., noise or crowd levels).

The goal of scheduling is to assign each patient one or more feasible appointment slots, ensuring that all hard constraints,such as clinical compatibility, doctor availability, accessibility, and resource limits, are strictly respected. Simultaneously, the system seeks to optimize soft objectives, including minimizing patient waiting times, adhering to expressed preferences, reducing clinician overload, and improving the overall utilization of healthcare infrastructure.

Within our ASP model, this task is cast as a Constraint Satisfaction Problem (CSP) enhanced with optimization criteria. Hard constraints prune invalid assignments, while weighted soft constraints guide the search toward the most desirable outcomes—delivering an automated, explainable, and ethically grounded scheduling strategy that adapts to the real-world complexities of modern healthcare delivery.

3.1 Formal Problem Statement

We formalize the medical appointment scheduling task as a constrained optimization problem defined as follows.

Given:

- A set P of patients, each characterized by a priority level $\text{priority}(p) \in \{\text{high}, \text{medium}, \text{low}\}$, clinical needs, and a set of preferences over time slots, doctors, clinics, and environmental conditions.
- A set D of doctors, each with specific expertise and capacity constraints.

- A set C of clinics, each with available time slots, accessibility features, and budgetary constraints.
- A set S of available time slots, distributed among clinics and doctors.

Constraints:

- Each patient must be assigned the required number of appointments matching their treatment plan.
- Assignments must respect hard constraints: accessibility requirements, clinic budgets, visit compatibility, and clinician availability.
- No double-booking is permitted for the same doctor, clinic, and time slot.

Objective: Find an assignment function $f : P \to (D \times C \times S)$ that:

1. Minimizes overall patient waiting times.
2. Maximizes satisfaction of patient preferences (preferred clinics, doctors, time windows, environmental conditions).
3. Ensures equitable workload distribution among doctors.
4. Minimizes total operational penalties (sensory mismatches, resource overuse).

This formalization naturally maps to an Answer Set Programming (ASP) model, where feasible assignments correspond to answer sets satisfying the constraints and optimization goals.

3.2 User Modeling and Personas

Effective appointment scheduling requires more than managing clinical data; it demands understanding patients as individuals with unique needs, preferences, and constraints. For this reason, we adopt **Blueprint Personas**, structured archetypes originally developed within European digital health initiatives (EIP on AHA) [11], which encapsulate real-world profiles combining clinical, social, cognitive, and behavioral attributes. Blueprint Personas enable a layer of abstraction between raw data and decision logic, allowing our system to reason over individualized scheduling needs in a flexible and human-centered way.

Patient personas: Each patient persona is a combination of health status, socio-environmental factors, and digital literacy; this model allows the system to reason over complex, individualized scheduling needs. Although digital literacy is currently included as part of the patient profile, it does not influence the scheduling logic directly. Its presence is intended for broader system integration purposes, for example, to guide interface selection (e.g., app-based vs. caregiver-mediated access) or to filter feasible options such as telemedicine. There are three different levels of patient complexity:

- **Generally healthy individuals**: prefer convenient scheduling (e.g., early mornings, nearby clinics) but have no strict clinical constraints.
- **People with chronic conditions**: require regular follow-ups, budget-sensitive allocation, and minimal waiting times; the scheduling must comply with multi-visit intervals and cost limits.
- **Patients with complex needs**: may express sensory sensitivities (e.g., noise, lighting), require accessible facilities, or depend on caregivers for travel; the system must respect personalized preferences and environmental constraints.

Key personal data identify patients and can express preferences for clinics, time slots, sensory conditions, and doctors. Travel distance is also considered, with a fixed value of 0 for Home Care and Telemedicine.

```
patient(p1, "Mario", "Rossi", "L'Aquila"). disabled(p1).
preference(p1, c3).
sensory_preference(p1, "noise").
doctor_preference(p1, "GP", "chronic_diseases", 10).
appointment_preference(p1, c3, 1850, 2000).
distance(p1, c3, 15).
```

Listing 1: Patient Profile with Preferences and Constraints

Clinician Personas: Equally important is the modeling of medical professionals; each clinician's persona describes, at least, their specialty, technological commitment, and operational challenges. This allows the system to anticipate realistic workload constraints and optimize care coordination.

For each available clinic, the **visit_type** represents the characteristics of the treatments offered in the clinics. Each visit is identified by: name, cost, and classification indicating its chronicity (0 = non-chronic, 1 = chronic), and the need for an in-person visit. Some visits require several sessions, with minimum and maximum intervals that must be followed:

```
visit_type(v1, "Cardiology", "Heart Attack", 0, 0, 0).
visit_cost(v1, 1000).
required_sessions(v1, 2).
session_interval(v1, 14, 28).
```

Listing 2: Visit Type Definition and Constraints

Clinic availability is defined by Unix timestamps, marking specific times for medical appointments. Each patient request includes an urgency level (1–3) and a preferred interval between sessions, thus enabling optimal scheduling based on personal needs and budget.

```
patient_interval(p1, v1, 21, 28).
need(p1, v1, 3).
availability(c1, v1, 1727308800).
```

Listing 3: Patient Needs Preferences and Clinic Availability

This modeling enables the system to reason over patient-specific constraints and ensures that solutions are medically appropriate and patient-centered. The personas also support scenario-based validation: simulated patients can be run through the solver to assess how well the system accommodates edge cases and vulnerable users.

3.3 Rules of inference

Inference rules in the ASP-based model derive intermediate knowledge from base facts, enhancing optimization beyond simple constraint filtering. These rules are not used as hard constraints or in choice rules, but instead serve to assign utility values to specific preferences, which are then aggregated in the optimization phase.

Patient preferences, such as favorite clinics, are translated into weighted indicators:

```
clinic_preference_effect(Patient, Clinic, 1) :-
    preference(Patient, Clinic).
clinic_preference_effect(Patient, Clinic, 0) :- not
```

```
    preference(Patient, Clinic).
```

Listing 4: Effect of Clinic Preference on Patient Assignment

Similarly, doctor preferences are evaluated based on specialization and years of experience:

```
doctor_preference_effect(Patient, Doctor, 1) :-
    patient(Patient, _, _, _),
    doctor(Doctor, _, _, _, _, Type),
    doctor_experience(Doctor, Specialization, YearsExperience),
    doctor_preference(Patient, Type, Specialization, RequiredYears),
    YearsExperience >= RequiredYears.
doctor_preference_effect(Patient, Doctor, 0) :-
    patient(Patient, _, _, _),
    doctor(Doctor, _, _, _, _, Type),
    doctor_experience(Doctor, Specialization, _),
    not doctor_preference(Patient, Type, Specialization, _).
doctor_preference_effect(Patient, Doctor, 0) :-
    patient(Patient, _, _, _),
    doctor(Doctor, _, _, _, _, Type),
    doctor_experience(Doctor, Specialization, YearsExperience),
    doctor_preference(Patient, Type, Specialization, RequiredYears),
    YearsExperience < RequiredYears.
```

Listing 5: Effect of Patient Preference on Doctor Assignment

The same logic is applied to time-slot preferences and sensory preferences, resulting in predicates such as `appointment_preference_effect/4` and `sensory_penalty/4`.

The **appointment_preference_effect/4** determines whether a given appointment time aligns with a patient's preferred time range: returns 1 if the time falls within the specified range (Start to End), and 0 otherwise. The time is transformed into a weighted score using hours and minutes to allow comparison with the preference interval; this value can be used in optimization to prioritize preferred time slots.

```
appointment_preference_effect(Patient, Time, Clinic, 1) :-
    patient(Patient, _, _, _),
    clinic(Clinic, _),
    availability(Clinic, _, _, Time),
    appointment_preference(Patient, _, Start, End),
    X = (((Time \ 86400) * 3600) * 100) + (((Time \ 3600) / 60) / 3) *
        5,
    X <= End, X >= Start.
appointment_preference_effect(Patient, Time, Clinic, 0) :-
    patient(Patient, _, _, _),
    clinic(Clinic, _),
    availability(Clinic, _, _, Time),
    appointment_preference(Patient, _, Start, End),
    X = (((Time \ 86400) * 3600) * 100) + (((Time \ 3600) / 60) / 3) *
        5,
    X > End.
appointment_preference_effect(Patient, Time, Clinic, 0) :-
    patient(Patient, _, _, _),
    clinic(Clinic, _),
    availability(Clinic, _, _, Time),
```

```
    appointment_preference(Patient, _, Start, End),
    X = (((Time \ 86400) * 3600) * 100) + (((Time \ 3600) / 60) / 3) *
        5,
    X < Start.
```

Listing 6: Effect of Time Preference on Appointment Scheduling

The **sensory_penalty/4** rule is useful in determining the gap between the environmental conditions of the clinic and the patient's expressed preferences.

```
sensory_penalty(Patient, Clinic, Time, Level):-
    sensory_preference(Patient, Type),
    environmental_condition(Clinic, Type, Level, Start, End),
    Time >= Start, Time <= End.
sensory_penality(Patient, Clinic, Time, 0):- not
    sensory_preference(Patient, _).
```

Listing 7: Sensory Penalty Based on Environmental Conditions

All inference rules defined above are used within the optimization statement shown below. Rather than enforcing constraints, they serve as soft indicators for optimization. The ASP solver minimizes travel distance and wait time while maximizing adherence to patient preferences. Penalties are added for environmental mismatches, and bonuses are applied when preferences are satisfied.

```
#minimize {
    (Distance * 10000) + WaitTime + (Penalty * 1000) -
    (ClinicPreference * 10000) -
    (DoctorPreference * 1000) -
    (AppointmentPreference * 1000) :
        distance(Patient, Clinic, Distance),
        appointment(Patient, Clinic, Doctor, _, Time),
        current_time(CurrentTime),
        WaitTime = Time - CurrentTime,
        sensory_penalty(Patient, Clinic, Time, Penalty),
        clinic_preference_effect(Patient, Clinic, ClinicPreference),
        doctor_preference_effect(Patient, Doctor, DoctorPreference),
        appointment_preference_effect(Patient, Time, Clinic,
        AppointmentPreference)
}.
```

Listing 8: Objective function minimizing cost based on preferences and constraints

This optimization function balances hard constraints (Section 3.4) with patient-centered utility values, enabling the system to produce high-quality, personalized, and ethically grounded schedules.

3.4 Constraints

Constraints define the logical rules that each solution generated by the ASP model must comply with. They ensure that appointments are allocated in accordance with clinical, personal, and logistical requirements.

First, each patient must be assigned exactly the number of appointments required for each treatment, according to their specific medical condition. This guarantees that treatments involving multiple sessions are correctly distributed across available time slots.

```
Sessions{ appointment(Patient, Clinic, Doctor,  Visit, Time) :
            availability(Clinic, Doctor, Visit, Time) }Sessions :-
need(Patient, Visit, _), required_sessions(Visit, Sessions).
```

Listing 9: Choice Rule for Appointment Allocation Based on Patient Needs

Moreover, to avoid scheduling conflicts, two patients can not be assigned the same clinic, visit type, and time slot simultaneously.

```
:- appointment(P1, Clinic, Doctor, Visit, Time), appointment(P2,
   Clinic, Doctor, Visit, Time), P1 != P2.
```

Listing 10: Constraint Preventing Double Booking of the Same Slot

Priority is also given to patients with high urgency, who must be scheduled in earlier slots, within a predefined temporal threshold.

```
:- needs(P1, Visit, Urg1), needs(P2, Visit, Urg2),
   Urg1 > Urg2, appointment(P1, Clinic, Doctor, Visit, Time1),
   appointment(P2, Clinic, Doctor, Visit, Time2),
   Time1 > Time2.
```

Listing 11: Constraint to Prioritize Urgent Visits

Accessibility needs are also addressed: patients with disabilities must only be assigned to clinics explicitly marked as accessible.

```
:- disabled(Patient), appointment(Patient, Clinic, _, _,_), not
   accessible(Clinic).
```

Listing 12: Constraint Preventing Assignment to Inaccessible Clinics

The model enforces clinic-side budget limits by ensuring that the total cost of scheduled chronic care appointments does not exceed each facility's allocated resources. This reflects real-world constraints in public healthcare, such as limits on subsidized services.

Patient-side financial constraints (e.g., ability to pay) are not currently modeled but are relevant in private or mixed healthcare systems. Their inclusion would require extending patient profiles and is planned as future work.

```
:- chronic_cost(Clinic, TotalCost),
   budget(Clinic, Budget), TotalCost > Budget.
```

Listing 13: Constraint to Respect Clinic Budget Limits

Some visits can occur via telemedicine or home care, but others require a doctor's physical presence and must be on-site; this constraint prevents assigning visits to incompatible clinic types.

```
:- appointment(Patient, Clinic, Doctor, Visit, Time),
   visit_type(Visit, _, _, _, 1, _),
   clinic(Clinic, "Home Care").
:- appointment(Patient, Clinic, Doctor, Visit, Time),
   visit_type(Visit, _, _, _, 1, _),
   clinic(Clinic, "Telemedicine").
:- appointment(Patient, Clinic, Doctor, Visit, Time),
```

```
    visit_type(Visit, _, _, _, _, 1),
    clinic(Clinic, "Telemedicine").
```

Listing 14: Constraints for Incompatible Visit Types and Clinic Modes

4 System Architecture

The system is implemented as a set of loosely coupled microservices that are organized to manage appointment requests and generate optimal schedules based on ASP logic. The system is structured in three main layers: User Interface, Business Logic, and Data Management, to ensure scalability, maintainability, and integration with existing healthcare infrastructures.

The **User Layer** includes both web interfaces (e.g., browser) and IoT devices, which may trigger appointment requests automatically based on health parameter thresholds.

The **Business Logic Layer** consists of two Flask-based microservices [24]:

- MS1: handles incoming API requests, user authentication (via JWT tokens [15]), and formats the data into ASP facts.
- MS2: collects facts into a batch queue and, every 60 seconds, compiles them into a complete ASP program; this program is solved using Clingo [13].

The **Data Layer** relies on a MySQL database, which maintains key information such as patient preferences, clinic features, appointment availability, environmental conditions, and budget constraints. All data interactions follow the ACID principles.

The system exposes several REST APIs for backend interaction; these include endpoints to submit appointment requests, check the status of ongoing bookings, and view up-to-date availability for each clinic. Asynchronous threading prevents overload during peak times, offering immediate user feedback and delayed result delivery.

Thanks to its layered architecture, centralized constraint management via ASP, and smooth integration with the Python backend, the system demonstrates effectiveness in handling complex scheduling scenarios. Moreover, the microservice-based structure allows individual components (e.g., solver, user interface, or database) to be updated or extended independently, without compromising the entire system. The full source code is publicly available online [1].

5 Experimental Evaluation

To illustrate the flexibility and robustness of the proposed ASP-based scheduling framework, we present three realistic scheduling scenarios inspired by real-world medical scheduling needs. The experiments were run on a machine with an Intel Core i7-9750HF CPU @ 2.60GHz, 16GB RAM, and Windows 11, using Clingo version 5.7.2. [2].

The scenarios presented in this section are based on synthetically generated data. Although not derived from real-world records, each example has been carefully designed to reflect realistic combinations of patient preferences, clinical requirements, and environmental conditions. All test inputs are encoded

[1] https://github.com/DawidPado/An-ASP-based-Solution-to-the-Medical-Appointment-Scheduling-Problem/tree/main

[2] https://github.com/potassco/clingo/releases/tag/v5.7.0

as ASP facts and are fully reproducible. These scenarios are intended to demonstrate the system's ethical and constraint-aware behavior under varied configurations.

Each scenario highlights the system's ability to handle key constraints such as accessibility, urgency, and personalized patient preferences. The optimization model described in Listing 8 plays a central role in guiding the scheduling decisions.

5.1 Scenario 1: High-Priority Patient with Accessibility Requirements

In this first scenario, a patient with a disability and a high-priority medical need requires a cardiology appointment. The patient, p1, must be scheduled for a visit type v1 (e.g., cardiology consultation) with urgency level 3. Due to the patient's condition, the appointment must be assigned only to clinics that are physically accessible. Listing 15 summarizes the key ASP facts for Scenario 1 in compressed form.

```
% Scenario 1: High-priority disabled patient (compressed facts)
doctor(m1, "Marco", "Bianchi", 52, "L'Aquila", "GP").
patient(p1, "Mario", "Rossi"). disabled(p1).
clinic(c3, "Clinic A"). ... clinic(c5, "Clinic C").
accessible(c3). accessible(c5).
distance(p1, c3, 15). distance(p1, c4, 5). distance(p1, c5, 20).
visit_type(v1, "Cardiology", "Hypertension", 0, 1, 1). need(p1, v1, 3).
availability(c3, m1, v1, 1727308800). availability(c4, m1, v1,
    1727481600).
availability(c5, m1, v1, 1727308800).
```

Listing 15: ASP facts for Scenario 1 – High-priority disabled patient

To ensure that the patient is only assigned to an accessible clinic, the Listing 12 constraint was used.

Outcome: The optimizer selected the earliest possible time in an accessible clinic with minimal travel distance. Since no sensory preferences were provided, the penalty was zero, and the solution was optimal with respect to both hard constraints and the minimization function. **Computation time:** 0.036s **Optimality:** Yes

5.2 Scenario 2 – Sensory Preferences

In this case, a patient expresses a preference for a low-noise environment. Although a closer clinic is available, the optimizer selects a more distant one that meets the environmental criteria:

```
% Patient and sensory preference
patient(p2, "Giulia", "Bianchi"). sensory_preference(p2, "light").
% Clinics and environmental conditions
clinic(c3, "Clinic A"). clinic(c4, "Clinic B").
environment_condition(c1, "light", 3, 1727480000, 1727489000).
environment_condition(c2, "light", 1, 1727480000, 1727489000).
distance(p2, c3, 10). distance(p2, c4, 25).
% Type of visit requested
visit_type(v2, "Orthopedics", "Kyphosis", 0, 1, 1).
visit_cost(v2, 1500). needs(p2, v2, 2).
% Slot availability
availability(c3, m1, v2, 1727481600). % 28.09.2024, 12:00
%...
```

Listing 16: ASP facts for Scenario 2 – Sensory preference for brightness

To evaluate the impact of environmental conditions on patient satisfaction, the same sensory penalty rules already introduced in Listing 7 were applied. These rules assess the alignment between patient sensory preferences and the clinic's environmental conditions, ensuring that discomfort factors (e.g., light, noise) are minimized during scheduling. As a result, the optimization function could correctly prioritize assignments that respect the patient's sensory needs.

Outcome: The patient was correctly assigned to Clinic B due to the lower sensory penalty. Despite the higher distance cost, the optimization logic prioritized sensory comfort, as expressed in the `sensory_penalty/4` term of the objective function. **Computation time:** 0.017s **Optimality:** Yes

5.3 Scenario 3 – Group Prioritization

Five patients request a cardiology visit at the same clinic. They differ in: **Urgency level**, **Preferences** (clinic, sensory), **Distance from the clinic**. The model must prioritize based on urgency while minimizing penalties.

```
% Patients and urgency levels
patient(p1, "Mario", "Rossi"). needs(p1, v1, 3). % High urgency
patient(p2, "Giulia", "Bianchi"). needs(p2, v1, 2). % Medium
%...
preference(p3, c3). preference(p3, c5).
% Distances from clinic
distance(p1, c3, 10). distance(p2, c3, 15).
%...
%Sensosy penalties
sensory_preference(p2, "light"). sensory_preference(p4, "light").
environmental_condition(c3, "light", 3, 1727481600, 1727488800).
% Requested visit type
visit_type(v1, "Cardiology", "Hypertension", 0).
% Visit type and availability
availability(c3, m1, v1, 1727481600). % 28.09.2024, 12:00
%...
```

Listing 17: ASP facts for Scenario 3 – Prioritization of 5 patients with limited slots

Outcome: All patients were correctly assigned respecting urgency, preferences, and minimizing overall cost. **Computation time:** 0.017s **Objective function value:** 1,610,000

Analysis of the result:

Mario (p1), with high urgency (3), was given top priority and assigned to the first available slot at 12:00. Giulia (p2) and Anna (p4), both with medium urgency (2), were scheduled in the subsequent slots, with Giulia prioritized due to her shorter travel distance. Luca (p3) and Paolo (p5), having low urgency (1), were assigned the last two slots. Luca, despite a slightly longer distance than Paolo, was scheduled first due to his clinic preference, which reduced overall cost.

The solution returned by Clingo has an optimization score of 1,610,000. This value is the result of the weighted sum defined in the `#minimize` directive and reflects the overall cost of the proposed schedule. It includes penalties for travel distance, delay, and sensory mismatch, as well as bonuses for satisfying individual preferences. Lower scores indicate better solutions.

5.4 Results Overview

The experimental evaluation demonstrates the effectiveness and adaptability of the ASP-based scheduling system across diverse real-world scenarios. Key findings highlight the system's ability to bal-

Patient	Urgency	Preferences	Slot Assigned	Penalty	Priority OK
p1	High (3)	Clinic	12:00	0	✓
p2	Medium	Sensory	13:00	0	✓
p3	Low	Clinic	15:00	0	✓
p4	Medium	Sensory	14:00	0	✓
p5	Low	–	16:00	0	✓

Table 1: Summary of patient assignments (Scenario 3)

ance competing objectives while adhering to complex constraints, ensuring both efficiency and patient-centered care.

Scenario	Constraints Respected	Preferences Respected	Time (s)	Optimal
Disabled patient	✓	✓	0.036	✓
Sensory preferences	✓	✓	0.017	✓
Multiple patients	✓	✓	0.017	✓

Table 2: Summary of Experimental Results

These experiments confirm the model's ability to handle:

- **Complex multi-constraint scheduling:** across all scenarios, the system strictly adhered to hard constraints. Invalid solutions (e.g., assigning a disabled patient to an inaccessible clinic) were automatically eliminated by the ASP solver.
- **Personalized preferences:** sensory preferences (Scenario 2) and clinic choices were respected even when conflicting with distance minimization, illustrating the system's capacity to tailor schedules to individual patient profiles.
- **Efficient optimization:** Computation times remained low ($<$0.04s) even for multi-patient scheduling (Scenario 3), confirming the system's suitability for real-time decision-making. The ASP solver consistently produced optimal and constraint-compliant solutions.
- **Fairness and Prioritization:** high-urgency patients (e.g., Scenario 1) were prioritized for the earliest available slots, minimizing waiting times. In group scheduling (Scenario 3), urgency levels dictated slot assignments, while preferences and penalties were optimized collectively, ensuring equitable resource distribution.

The ASP-based system demonstrates robustness, flexibility, and suitability for real-world healthcare scheduling tasks. The results validate the system's applicability in healthcare settings, where rapid adjustments to cancellations, emergencies, or fluctuating resource availability are essential. Across all test cases, the ASP solver produced optimal schedules in under 0.04 seconds, confirming the feasibility of real-time decision-making even with complex constraints.

Although the scenarios presented are illustrative, the ASP encoding is designed for larger-scale scheduling with controlled grounding. Since most rules are patient- and slot-specific, without cross-patient aggregates, the number of grounded atoms grows with $O(p \cdot c \cdot s)$ in the worst case—where p is the number of patients, c clinics, and s slots.

In practice, grounding is reduced by filtering irrelevant clinics and time slots per patient. In tests conducted with a custom Python generator, using 50 patients, 6 clinics, and 500 slots, the grounded atom

count stayed below 10,000, and solving time was under 0.1 seconds. Performance remained stable with low contention and diversified visit types.

While a full scalability study is planned, these preliminary tests support the claim that the model scales linearly in realistic conditions.

6 Conclusion and Future Work

Medical appointment scheduling remains a critical bottleneck in healthcare operations, where balancing patient needs with resource optimization presents persistent challenges. This work introduces an ASP-based framework that systematically addresses these challenges through fully declarative constraint modeling and multi-objective optimization. The framework demonstrates rapid solution times, consistently generating optimal schedules in under 0.04 seconds, validating its feasibility for real-time healthcare settings.

By integrating Blueprint Personas, patient priorities, sensory preferences, accessibility requirements, and clinic budgets, the framework supports personalized care pathways for vulnerable populations, such as individuals with chronic conditions or sensory sensitivities. These results bridge the gap between theoretical optimization models and practical healthcare needs, offering a scalable, transparent, and ethically grounded alternative to heuristic-driven approaches.

Although the current evaluation focuses on constrained scenarios, further experimentation is required to assess performance under extreme loads (e.g., hundreds of concurrent scheduling requests) and in highly dynamic environments (e.g., frequent emergencies). Additionally, incorporating predictive analytics for no-show estimation could further enhance scheduling robustness. Future work also includes extending the framework to support neurodivergent patients during urban commuting, broadening its societal impact.

In summary, the ASP-based approach delivers a flexible, efficient, and ethically aligned solution to appointment scheduling, addressing critical gaps in existing systems while promoting both clinical effectiveness and patient-centered care.

Scalability and Future Directions. Our experimental results demonstrate that the framework effectively handles real-world scheduling complexity with rapid solution times. Future research will focus on scaling the system to accommodate larger instances involving hundreds or thousands of scheduling requests. Planned enhancements include incremental ASP solving, solver portfolio techniques, and hierarchical scheduling decomposition to ensure computational efficiency in high-demand environments.

Potential for Generalization. Although the present system targets medical appointment scheduling, the underlying ASP-based framework is domain-agnostic and adaptable. By modifying input facts and constraint encodings, the same architecture can be applied to a wide range of scheduling tasks, including:

- Workforce rostering in healthcare, manufacturing, and service industries.
- School and university course timetabling, incorporating preferences and resource constraints.
- Facility booking and event scheduling with logistical optimization.

This generalizability highlights the broader applicability of ASP-based declarative optimization in dynamic, real-world planning domains.

Aknowledgments

Research partially supported by the PNRR Project CUP E13C24000430006"Enhanced Network of intelligent Agents for Building Livable Environments - ENABLE", and by PRIN 2022 CUP E53D23007850001 Project TrustPACTX - Design of the Hybrid Society Humans-Autonomous Systems: Architecture, Trustworthiness, Trust, EthiCs, and EXplainability (the case of Patient Care), and by PRIN PNNR CUP E53D23016270001 ADVISOR - ADaptiVe legIble robotS for trustwORthy health coaching.

References

[1] Ali Ala, Fawaz E Alsaadi, Mohsen Ahmadi & Seyedali Mirjalili (2021): *Optimization of an appointment scheduling problem for healthcare systems based on the quality of fairness service using whale optimization algorithm and NSGA-II*. Scientific Reports 11(1), p. 19816, doi:10.1038/s41598-021-98851-7.

[2] Ali Ala & Feng Chen (2022): *Appointment scheduling problem in complexity systems of the healthcare services: A comprehensive review*. Journal of Healthcare Engineering 2022(1), p. 5819813, doi:10.1155/2022/5819813.

[3] Ali Ala, Vladimir Simic, Dragan Pamucar & Erfan Babaee Tirkolaee (2022): *Appointment scheduling problem under fairness policy in healthcare services: fuzzy ant lion optimizer*. Expert systems with applications 207, p. 117949, doi:10.1016/j.eswa.2022.117949.

[4] Mahmoud H Alrefaei & Ali Diabat (2015): *Modelling and optimization of outpatient appointment scheduling*. RAIRO-Operations Research-Recherche Opérationnelle 49(3), pp. 435–450, doi:10.1051/ro/2014041.

[5] Mario Alviano, Carmine Dodaro & Marco Maratea (2019): *Nurse (re) scheduling via answer set programming*. Intelligenza Artificiale 12(2), pp. 109–124, doi:10.3233/ia-170030.

[6] Chitta Baral (2003): *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press.

[7] Jürgen Branke, Su Nguyen, Christoph W Pickardt & Mengjie Zhang (2015): *Automated design of production scheduling heuristics: A review*. IEEE Transactions on Evolutionary Computation 20(1), pp. 110–124, doi:10.26686/wgtn.13151315.

[8] Paola Cappanera, Marco Gavanelli, Maddalena Nonato & Marco Roma (2023): *Logic-based Benders decomposition in answer set programming for chronic outpatients scheduling*. Theory and Practice of Logic Programming 23(4), pp. 848–864, doi:10.1017/s147106842300025x.

[9] Stefania Costantini & Andrea Formisano (2010): *Answer set programming with resources*. Journal of Logic and Computation 20(2), pp. 533–571, doi:10.1093/logcom/exn082.

[10] Carmine Dodaro, Giuseppe Galatà, Martin Gebser, Marco Maratea, Cinzia Marte, Marco Mochi & Marco Scanu (2024): *Operating room scheduling via answer set programming: Improved encoding and test on real data*. Journal of Logic and Computation 34(8), pp. 1556–1579, doi:10.1093/logcom/exae041.

[11] Commission E. (2024): *European Innovation Partnership on Active and Healthy Ageing*. https://ec.europa.eu/eip/ageing/home_en.html. Accessed: 2024-09-23.

[12] Xuanzhu Fan, Jiafu Tang, Chongjun Yan, Hainan Guo & Zhongfa Cao (2021): *Outpatient appointment scheduling problem considering patient selection behavior: data modeling and simulation optimization*. Journal of Combinatorial Optimization 42, pp. 677–699, doi:10.1007/s10878-019-00487-x.

[13] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2014): *Clingo= ASP+ control: Preliminary report*. arXiv preprint arXiv:1405.3694. Available at https://arxiv.org/abs/1405.3694.

[14] Diwakar Gupta & Brian Denton (2008): *Appointment scheduling in health care: Challenges and opportunities*. IIE transactions 40(9), pp. 800–819, doi:10.1080/07408170802165880.

[15] Muhamad Haekal et al. (2016): *Token-based authentication using JSON web token on SIKASIR RESTful web service*. In: *2016 International Conference on Informatics and Computing (ICIC)*, IEEE, pp. 175–179, doi:10.1109/iac.2016.7905711.

[16] Markus Hannebauer & Sebastian Müller (2001): *Distributed constraint optimization for medical appointment scheduling*. In: *Proceedings of the fifth international conference on Autonomous agents*, pp. 139–140, doi:10.1145/375735.376026.

[17] Stavros Kanias (2023): *An AI Approach to Large-Scale Medical Appointment (Re) Scheduling Using ASP.* In: *ENIGMA@ KR*, pp. 35–42.

[18] Alex Kuiper, Jeroen de Mast & Michel Mandjes (2021): *The problem of appointment scheduling in outpatient clinics: a multiple case study of clinical practice*. *Omega* 98, p. 102122, doi:10.1016/j.omega.2019.102122.

[19] Yong-Hong Kuo, Hari Balasubramanian & Yan Chen (2020): *Medical appointment overbooking and optimal scheduling: tradeoffs between schedule efficiency and accessibility to service*. *Flexible Services and Manufacturing Journal* 32, pp. 72–101, doi:10.1007/s10696-019-09336-z.

[20] Vladimir Lifschitz (2008): *What is answer set programming?* In Frank van Harmelen, Vladimir Lifschitz & Bruce Porter, editors: *Handbook of Knowledge Representation*, *Foundations of Artificial Intelligence* 3, Elsevier, pp. 889–910, doi:10.1016/S1574-6526(07)03027-1.

[21] Vladimir Lifschitz (2019): *Answer set programming*. 3, Springer Cham, doi:10.1007/978-3-030-24658-7.

[22] Ministero del Lavoro, della Salute e delle Politiche Sociali (2009): *Sistema CUP: Linee guida nazionali*. https://www.salute.gov.it/imgs/C_17_pubblicazioni_1577_allegato.pdf. Accessed: 2025-04-02.

[23] Andrea Monaldini, Alina Vozna & Stefania Costantini (2024): *Blueprint Personas in Digital Health Transformation*. In: *Workshop HC@ AIxIA 2024*, pp. 40–49. Available at https://ceur-ws.org/Vol-3880/paper4.pdf.

[24] Sam Newman (2021): *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.".

[25] Tiantian Niu, Bingyin Lei, Li Guo, Shu Fang, Qihang Li, Bingrui Gao, Li Yang & Kaiye Gao (2023): *A review of optimization studies for system appointment scheduling*. *Axioms* 13(1), p. 16, doi:10.3390/axioms13010016.

[26] Farid Nouioua & Pascal Nicolas (2006): *Using answer set programming in an inference-based approach to natural language semantics*. *arXiv preprint cs/0607088*. Available at https://arxiv.org/abs/cs/0607088.

[27] Huaxin Qiu, Dujuan Wang, Yanzhang Wang & Yunqiang Yin (2019): *MRI appointment scheduling with uncertain examination time*. *Journal of Combinatorial Optimization* 37, pp. 62–82, doi:10.1007/s10878-017-0210-5.

FÆRDXEL: An Expert System for Danish Traffic Law

Luís Cruz-Filipe
IT University of Copenhagen
Copenhagen, Denmark
luic@itu.dk

Jonas Vistrup
University of Southern Denmark
Odense, Denmark
vistrup@imada.sdu.dk

We present FÆRDXEL, a tool for symbolic reasoning in the domain of Danish traffic law. FÆRDXEL combines techniques from logic programming with a novel interface that allows users to navigate through its reasoning process, thereby ensuring the system's explainability. Towards the goal of better understanding the value of FÆRDXEL, two evaluations of the system have been performed: (1) An empirical evaluation showing that for a selection of court cases, the conclusions of FÆRDXEL align with those of Danish judges. (2) A qualitative evaluation from legal experts indicating that this work has potential to become a foundation for real-world AI tools supporting professionals in the Danish legal sector.

1 Introduction

The release of ChatGPT in late 2022 sparked a global desire to integrate data-driven Artificial Intelligence (AI), based on sophisticated modern machine learning techniques, into as many domains as possible. However, some domains seem resistant to integrating this type of AI into their workflow. For some domains, machine learning is not a suitable approach, and for others, there is an established scepticism to the technology.

The domain of law exhibits both types of resistance. First, in many legal systems, there is a mandate for an explanation of judgments [2], and it is unclear from precedent whether the explanation has to be the specific reason for the judgment or just a possible reason behind the judgment. It can be argued that machine learning techniques, in the form of generative and deep neural networks, can only provide the latter. Second, legal systems are traditionally based on discussion and argumentation, which clashes with AI systems that only provide definitive answers without justification.

One possible approach to overcome these issues is to revert to more traditional forms of AI, in the form of expert systems. Expert systems provide symbolic reasoning using a knowledge base and a theoretically understood, sound inference algorithm. They also can provide a trace of their inference process, allowing us to scrutinize the reasoning behind its answers and, in some sense, argue with the system. However, the knowledge base must typically be tailored for the specific application in mind, so the system can reason within the domain of that application similar to how an expert might reason. This makes building expert systems very time-consuming. Another known limitation is that expert systems typically suffer from an interface problem, and using them often requires dedicated training – limiting their widespread usage [8].

This work. We present FÆRDXEL, an expert system designed to assist legal experts working with cases regarding possible violations of Danish traffic law. Specifically, FÆRDXEL looks for arguments supporting that a defendant has broken specific paragraphs of the Danish traffic law, both taking into account the law itself and similar cases that may be relevant. The goal of FÆRDXEL is not to make any decisions, but rather to provide meaningful and extensive input to those agents who will actually make

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 383–396, doi:10.4204/EPTCS.439.26

© L. Cruz-Filipe & J. Vistrup
This work is licensed under the
Creative Commons Attribution License.

the decisions – in the same way that a legal aid would. FÆRDXEL is based on classic principles from logic programming. Its knowledge base is manually written in an enriched version of Datalog, and it uses SLD-resolution as its inference system. Every fact and rule in FÆRDXEL's knowledge base comes with a schematic translation into natural language. FÆRDXEL answers questions by creating SLD-refutations, which are used to build a tree-like structure of possible explanations for each answer. FÆRDXEL also provides an interface for navigating through these explanations in a user-friendly way using the provided translations. Thus, users can examine FÆRDXEL's arguments and conclusions without understanding its internal language – this knowledge is only required to enter new information to FÆRDXEL, e.g., to edit facts about a specific case.

FÆRDXEL was developed as a first step in an interdisciplinary collaboration to understand the potential usage of expert systems in the legal domain, at a time when data-driven AI (and especially generative AI) is opening up new possibilities. Ultimately, our goal is to develop systems that can interact fully in natural language, and reason over a variety of domains. We chose to focus initially on traffic law because it is a simple domain, with clear-cut rules, making the logic formulation technically simpler – and allowing us to focus quicker on understanding how best to create and provide explanations.

Structure of the paper. Section 2 contains the relevant related work for our development. In Section 3, we describe the language and inference system of FÆRDXEL, and how they differ from standard tools. Section 4 describes in more detail the process of converting laws into logical rules. The mechanism underlying the creation of explanations by FÆRDXEL is described in Section 5, and Section 6 focuses on the system's interface. We have also undertaken both a small, qualitative evaluation of potential social value of FÆRDXEL and an empirical evaluation of the correctness of its answer, both of which we describe in Section 7. Finally, Section 8 concludes and points out directions for future work. The system implementation is available at https://doi.org/10.5281/zenodo.15626892.

2 Related Work

Research into expert systems and applications flourished in the 1980s including applications to the legal domain. However, this direction of research slowed down in the 2000s, and since 2010, to the best of our knowledge, there have been few publications in the domain of expert systems for law [8]. An interesting exception, from 2016, is an article on an expert system for Islam inheritance law [1].

Development of expert system applications for the legal domain go as far back to 1977's *TAXMAN* [20], an expert system for answering questions about US taxation for corporate reorganization. This work paved the way for several subsequent expert system, such as: *CHIRON* [26], for planning actions that could decrease income tax according to US law; *Split up* [32], which dealt with Australian law on property division after divorce; and *ASHSD-II* [22], focusing on US law on property division after divorce.

All of these applications focus on very specific subdomains within already specific legal domains: In domain of US taxation, *TAXMAN* on the subdomain of corporate reorganization and *CHIRON* on the subdomain of income tax. Reversely, both *Split up* and *ASHSD-II* on the subdomain of property division, but in different overarching domains, as *Split up* deals with Australian divorce law and *ASHSD-II* with US divorce law. In contrast, FÆRDXEL reasons on the whole of Danish traffic law, instead of being restricted to a subdomain like parking laws or driving laws. The tool is also designed such that additional laws can be added to its knowledge.

Other applications have aimed at a broader approach by allowing for reasoning within arbitrary legal domains. *SHYSTER* [23] is an expert system for providing and arguing similarities and differences between cases. It provided a framework for inputting knowledge about any legal case in any domain. In contrast to our work, however, *SHYSTER* did not contain knowledge about laws and was unable to reason about legal consequence. This was rectified by combining *SHYSTER* with a knowledge base. The resulting system *SHYSTER-MYCIN* [21] was thus able to reason using concrete laws, but again at the price of restricting its scope – in this case, to the subdomain of copyright law within Australian property law. In contrast, FÆRDXEL is able to reason on the entirety of Danish Traffic law.

Research on expert system applications in other domains has also been lacking in the last decade due to the increasing success of machine learning based systems. A notable exception is the medical diagnostic domain, where research into expert system applications has seen a low but steady output, demonstrate the utility of symbolic reasoning domains where the interpretability of results is critical, an important property within law. Examples include systems for diagnosing chronic heart failure using logic programming [3] and for diagnosing ischemic heart disease using the results of certain tests [12]. More recently, an expert system provided non-experts with common diagnoses of their ankle problems [7]. The user would answer a series of questions and the system would either reach a conclusion or decide that a doctor should be consulted. A questionnaire was also used to diagnose mental problems [14].

Since the release of ChatGPT, there has also been a significant increase in AI based commercial tools to assist the daily work of legal experts [27]. To the best of our knowledge, all recently released commercial AI legal tools are based on generative AI, primarily LLMs (Large Language Models) [5, 13, 15, 19, 24]. LLMs are able to generate arguments to a much wider degree than expert systems, since they can provide additional arguments based on the context and meaning of words [4]. However, these LLMs are trained on such large datasets that these sets likely contain undesired bias, which is transferred to the LLM during training [29]. This bias can be introduced when the LLM is interpreting facts, deriving conclusions or creating justifications.

To remove undesired bias, the LLM will often go through rounds of post-training called alignment [16, 31], e.g., reinforcement learning from human feedback [30], but it is currently unclear if these methods removes bias or only removes some expressions of bias [10]. In fact, it could well be the case that tools for law based on LLMs could provide well-hidden biased answers to legal queries: an LLM, which sequentially generates its output, could first generate a biased answers to a legal question and then generate a justification for said answer, selectively generating certain non-biased justifications, while leaving out the biased justifications for the answer, as the alignment has reduced the likelihood of the biased justification being presented. In that case, biased answers would be harder to detect after alignment, leading to undesired effects. FÆRDXEL avoids this issue by generating full explanations for answers before deducing answers and subsequently show them to the user.

3 Background

This section covers the necessary knowledge needed to understand the remainder of our work. We discuss the logical language Datalog, which we use to represent the knowledge inherent in Danish traffic law, and briefly summarize the key concepts of SLD-resolution, the inference algorithm used by FÆRDXEL. Lastly, we provide a brief description of the Danish court system.

3.1 Datalog

Datalog is the negation-free and function-free fragment of Prolog [9]. The basic formulas of Datalog are called *atoms*. Each atom describes a relation between objects. All atoms are on the form of $P(t_1, t_2, \ldots, t_n)$ where P is a *predicate* describing the type of relation and t_1 through t_n are all *terms* given as arguments for the predicate P. A term can either be a *constant* or a *variable*. A constant denotes a specific known object, and a variable denotes some unknown object. A *substitution* is a function mapping variables to terms, and the result of *applying* a substitution to a formula is computed by simultaneously replacing each variable by its value according to the substitution. The result of applying a substitution θ to a formula α is written $\alpha\theta$.

A Datalog *program* is a set of *rules* (also called clauses). Each rule is in the format

$$\alpha \leftarrow \beta_1, \beta_2, \ldots, \beta_n$$

where α and β_1 through β_n are atoms. The semantic understanding of the rule is if β_1 through β_n (the premises) holds then α (the conclusion) holds. If $n = 0$ then the rule simply states that α holds, and is called a *fact*.

3.2 SLD-resolution

SLD-resolution is an algorithm to compute *answers* to *queries* over Datalog programs. A query is a set of atoms, and an answer is a substitution of the variables in the query which makes the query a logical consequence of the program.

SLD-resolution uses *SLD-derivations*, which work on *goals*. A goal is a formula of the form $\alpha_1, \ldots, \alpha_n$. The basic ingredients for constructing SLD-derivations are *unifiers* and *resolution steps*.

A substitution θ *unifies* two formulas α and β if $\alpha\theta$ is the same as $\beta\theta$. Finding whether such a substitution exists for given α and β is known to be decidable, and there is a standard *unification algorithm* that computes a specific unifier of any two formulas, if one exists.

Resolution chooses one atom α in a goal G and a rule r from the program such that α can be unified with the conclusion of r. The result is a new goal G', which is obtained by applying the unifier of α and the head of r to both G and r, and replacing $\alpha\theta$ in the result with the premise of $r\theta$. (In particular, the selected atom is simply removed if r is a fact.) An SLD-derivation is a finite sequence of goals $G_0, \ldots, G_n$ where G_0 is the query and G_{i+1} is obtained from G_i by a resolution step.

An SLD-derivation also computes a substitution, which is obtained by composing the substitutions computed at each resolution step and restricting the result to the variables in the query. If G_n is a goal with no atoms, the SLD-derivation is called an *SLD-refutation*. Soundness of SLD-resolution guarantees that the substitution computed by an SLD-refutation starting with a goal $\leftarrow Q$ is an answer to the query Q, and completion of SLD-resolution guarantees that every answer to Q can be obtained as a composition $\theta\sigma$, where θ is computed by an SLD-refutation with starting goal $\leftarrow Q$.

A more detailed discussion of SLD-resolution can be found in e.g. [17].

3.3 Specifics of the Danish court system

In the Danish court system, the vast majority of cases is settled directly by sitting judges. Defendants are only allowed to present their case in front of a jury in extreme criminal scenarios where the prosecution is demanding a prison sentence of more than 6 years. These cases typically involve murder, rape, brutal robbery or extensive arson.

Since the judges are the arbitrators, court cases typically take on a procedural tone, and are usually predominantly technical. Therefore, an expert system designed to be used in this setting should focus on the technical aspects of the law – both what legislation says, and what can be inferred from similar cases. This is also where many resources in the Danish legal system are used: both judges and lawyers are assisted by teams who gather information about relevant laws and cases, and help developing possible argumentations.

4 Translation from Law to Rules

Reasoning systems are usually built from two components: a *knowledge base*, containing information about the reasoning domain, and a *reasoning engine* that can derive conclusions from the knowledge base. In this section we focus on how FÆRDXEL's knowledge base of 2579 rules was constructed. Our goal was that this knowledge base should both have internal consistency, and be updatable in the future. Therefore we present the overarching structure of the translation from natural knowledge to FÆRDXEL's system knowledge.

Laws are typically formulated in the form of rules that must be upheld; however, legal argumentation is more often focused on whether someone has broken the law – this is also the task that FÆRDXEL is expected to address. To this end, we view the law as specifying pairs containing a situation and a prohibited action in that situation. For each paragraph of the Danish traffic law, we formalize in Datalog both the situation the law is specifying, and the prohibited action.

Formalization can be done in a myriad of ways, so to ensure a consensus between rules in the knowledge base, some general principles have been developed and adhered to. The main principle is one of 'reification', which is the philosophical idea of describing abstract concepts as objects. In the context of FÆRDXEL we first define objects, either physical or abstract, through single argument predicates, then use generic predicates named by function words (`Has`, `On`, `With`) to describe the relation between the objects, and end by using specific predicates with few arguments (at most 4) to describe relations typically relevant only to the paragraph under consideration.

In addition, there is some information hidden in the context, and the applicable case law. To ensure the inclusion of this information, we use a standard reference [28] containing an in-depth understanding of Danish traffic law and relevant case law as provided by a group of legal experts – this is also an information source used by professionals. Furthermore, to capture the knowledge of the law for events with causes and effects, the system must understand that something can be true at on time, yet false at another time. Therefore all predicates $P(t_1,\ldots,t_n)$ have a time component τ. The system has two ways of expressing the time component of a predicates, either specifying the relation holds at time τ by adding it as another argument at the end, $P(t_1,\ldots,t_n,\tau)$, or by omitting it, $P(t_1,\ldots,t_n)$, to specify that the relation holds at all times.[1] These steps can be summarized, for each part of the law, as:

1) Identify the situation and prohibited action using standard reference [28] and relevant case law.
2) Formulate as much of the situation and action using previously defined predicates.
3) Try to formulate remaining parts of the situation and action using function words between abstract or non-abstract objects.
4) Formulate any remaining parts as specific predicates.

[1] Syntactically, this makes our language similar to Temporal Datalog [25]. However, we do not treat the temporal argument in any special way, so we can still remain in the realm of pure Datalog.

Often the law specifies similar situations where the same action is prohibited, or similar actions that are prohibited in the same situation. Since each situation/action combo is translated into a rule, this leads to a combinatorial explosion where a single part of a paragraph in some cases generate over 50 rules that only slightly differ. To make it easier to write such rules and minimize the risk of errors in translation, we include some syntactic sugar and allow premises of rules to include non-atomic formulas using disjunctions (`\/`) and conjunctions (`/\`). These formulas are then converted into Conjunctive Normal Form (CNF) using standard algorithms, yielding standard Datalog rules – so that the knowledge base is still pure Datalog.

Example 1. *§4.1 of the Danish traffic law can be translated into the following English:*

```
Road Users must follow the instructions given by traffic signs,
  markings on road, traffic lights or a similar method.
```

We use the steps for translating to derive a Datalog translation of §4.1. Step 1) Using the reference material [28], the situation become a road user on a road with a traffic sign, marking, traffic light or similar, which are providing an instruction. The action is the road user not following the instruction. Step 2) Let us assume `BrokenLaw` *and* `RoadUser` *are previously defined predicates that can be reused. Step 3) We can find the following objects:* `Instruction`*,* `Road`*,* `Sign`*,* `Marking` *and* `TrafficLight`*. We can also identify the following function words:* `Has` *and* `On`*Step 4) As we are lacking a way to specify similarity and not following an instruction, the following specific predicates are defined:* `SimilarTo` *and* `NotFollowing`*. Using the language we have specified,* §4.1 *can be translated into:*

```
1 BrokenLaw(P,§4.1,T) <- RoadUser(P,T), Instruction(I), Road(R),
2   Sign(Z) \/ Marking(Z) \/ TrafficLight(Z) \/ SimilarTo(Z,a_sign)
3   \/ SimilarTo(Z,road_marking) \/ SimilarTo(Z,traffic_light),
4   Has(Z,I), On(P,R,T), On(Z,R), NotFollowing(P,I,T)
```

where `P` *is a person,* `I` *an instruction,* `R` *a road,* `Z` *the object giving* `I` *and* `T` *is the time component.* ◁

For presentation, it is essential that all predicates can be translated into natural language. Therefore, FÆRDXEL's knowledge base also includes a set of definitions of natural-language variants for each predicate. This is the only place where temporal arguments get special treatment, as they are handled uniformly for all predicates.

Example 2. *The predicates used to implement §4.1 are associated with the following natural language translations.*

```
RoadUser(P):P is a road user; Instruction(I):I is an instruction;
Road(R):R is a road; Sign(S):S is a sign; Marking(M):M is a marking;
TrafficLight(Z):Z is a trafficlight; On(A,B):A is on B; Has(A,B):A has B;
NotFollowing(P,I):P isn't following I; BrokenLaw(P,L):P has broken L;
```

Temporal arguments are omitted in these translations. If a predicate includes a temporal argument `T`*, then* `at T` *is appended to its translation.* ◁

5 Inference System and Explainability

In this section, we discuss how the SLD-refutations created by the reasoning engine of FÆRDXEL are compactly stored to provide explanations for the system's conclusions.

Each explanation is stored in a tree format, as in Figure 1, where each node is an atom, and the root node is the original query. All other nodes are separated into two types: explanations and reasons. The

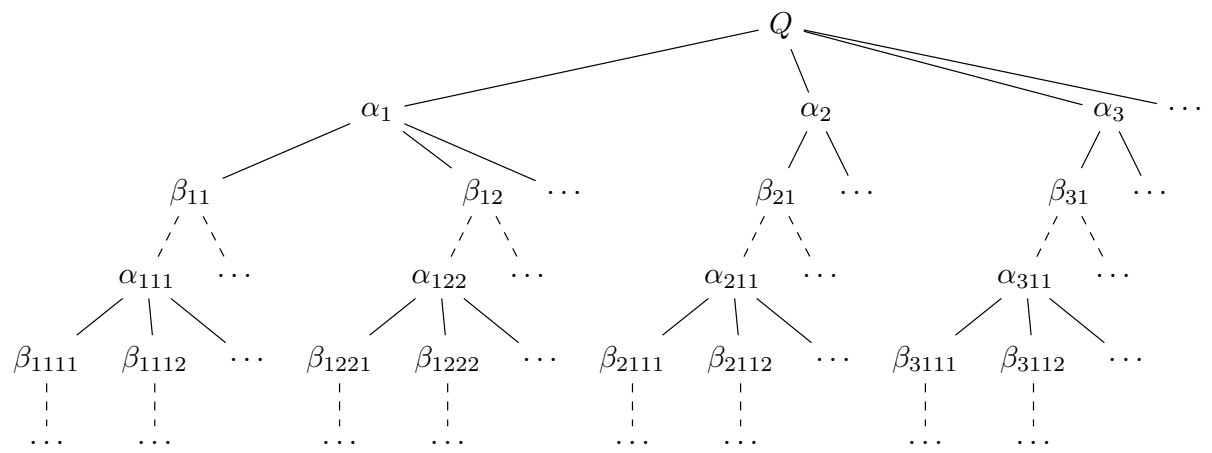


Figure 1: Explanation in FÆRDXEL. α-nodes are explanation-nodes and β-nodes are reasons-nodes.

type of nodes alternate between layers, i.e., the child of an explanation-node is a reason-node and vice versa. The children of the root node are explanation-nodes.

For each explanation-node α' with parent node β', there exists a clause $\alpha \leftarrow \beta_1, \beta_2, \ldots, \beta_n$, such that $\beta'\theta = \alpha\theta = \alpha'$ for some substitution θ, i.e., the parent node β' unifies with the head of the clause α resulting in α'. Its children are the reason-nodes $\beta_1\theta$, $\beta_2\theta$, …, $\beta_n\theta$.

An explanation-node should be interpreted as *an* explanation for why the parent node's atom is valid and its children's atoms should be interpreted as the atoms that *all* must be valid for an explanation to be valid (this structure is a a variant of the classic AND/OR-tree).

The tree is constructed using only valid SLD-refutations, which ensures that the tree is finite and that the atom of every node is valid [2].

Example 3. *To illustrate the process we now introduce a case that uses the paragraphs of the law encoded in the previous section. This case is also used for showcasing the interface in Section 6.*

The case regards a person overtaking a patrol officer on a road where passing is not allowed [6]. The overtaking occurs at 15:15, and there are both a sign and road markings stating that passing is not allowed. For privacy reasons, certain details of the case are anonymized, so the person is referred to as the `driver`*, their car as* `car`*, and the road as* `road`*.*

The facts of the cases can be encoded as:

```
Driving(driver,car), Road(road), Sign(sign), On(sign,road),
Marking(line), Instruction(no_passing), Has(sign,no_passing),
Has(line,no_passing), On(line,road), On(car,road,15:15),
On(driver,road,15:15), NotFollowing(driver,no_passing,15:15)
```

In addition to the encoding of §4.1 *from the previous section, assume our knowledge base also includes two rules derived from* §2.25 *of the Danish traffic law, which define the predicate* `RoadUser`*:*

```
RoadUser(P,T) <- On(P,R,T), Road(R)
RoadUser(P,T) <- Driving(P,C), On(C,R,T), Road(R)
```

To find out if any laws have been broken, we send the query `BrokenLaw(P,X,T)` *to* FÆRDXEL*, which first generates all possible SLD-refutations using SLD-resolution. In this case, four refutation are created*

[2] Validity, when the atom contains variables, requires that at least one assignment of the variables is valid

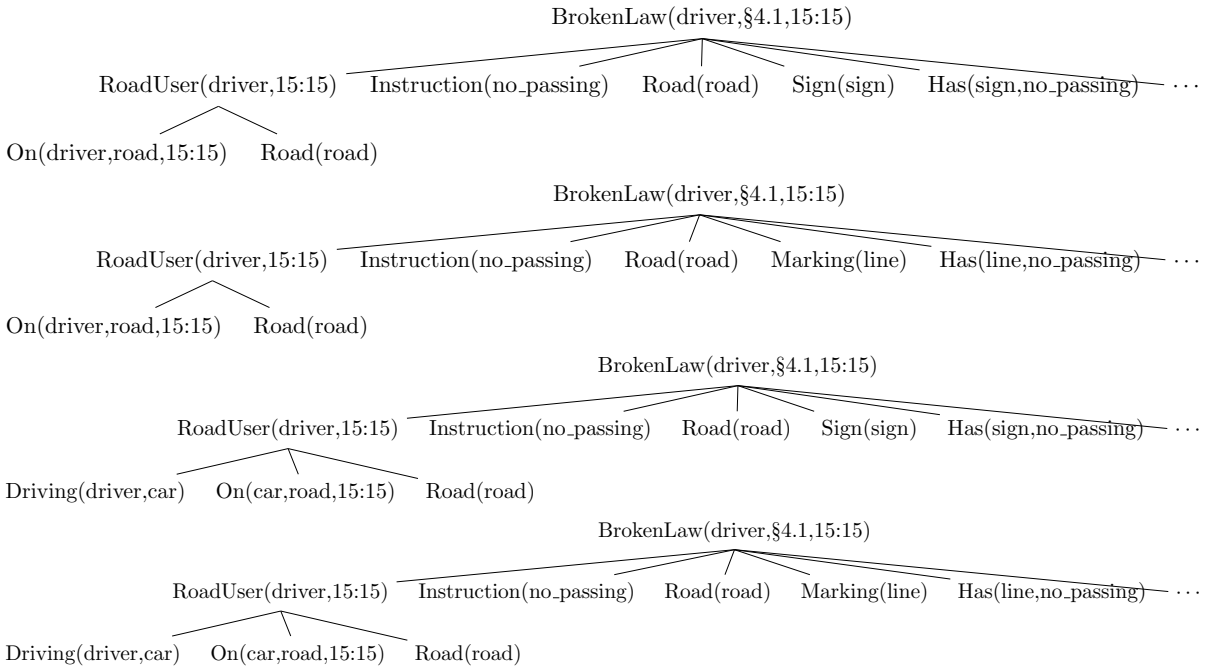


Figure 2: Four proof trees created by querying `BrokenLaw(P,X,T)` on Example 3.

together with their proof tree (Figure 2). These proof trees are then combined to create an explanation (Figure 3).

6 Interface

We now showcase the interface of FÆRDXEL from the perspective of a user, using the case introduced in the previous section (Section 5). Our focus is on displaying explanations in an understandable way and allowing users to navigate them. Currently our prototype has only a minimalistic interface providing these capabilities.

We do not include actual screenshots, but instead redrawings of the interface. This is done both to exacerbate details and because the actual interface is in Danish. For illustration, we use the case presented in the previous section.

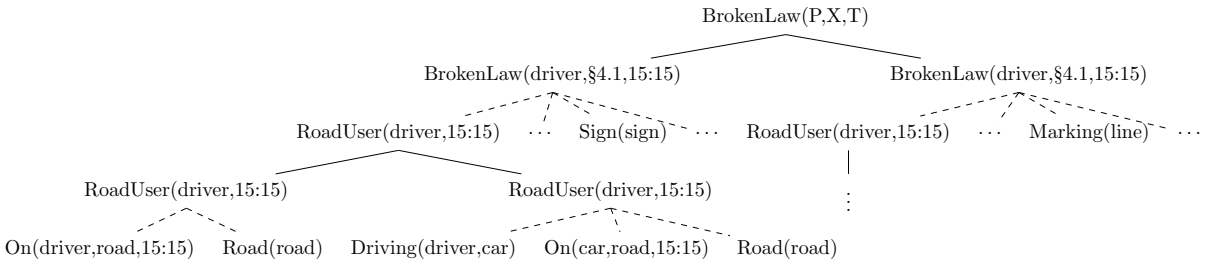
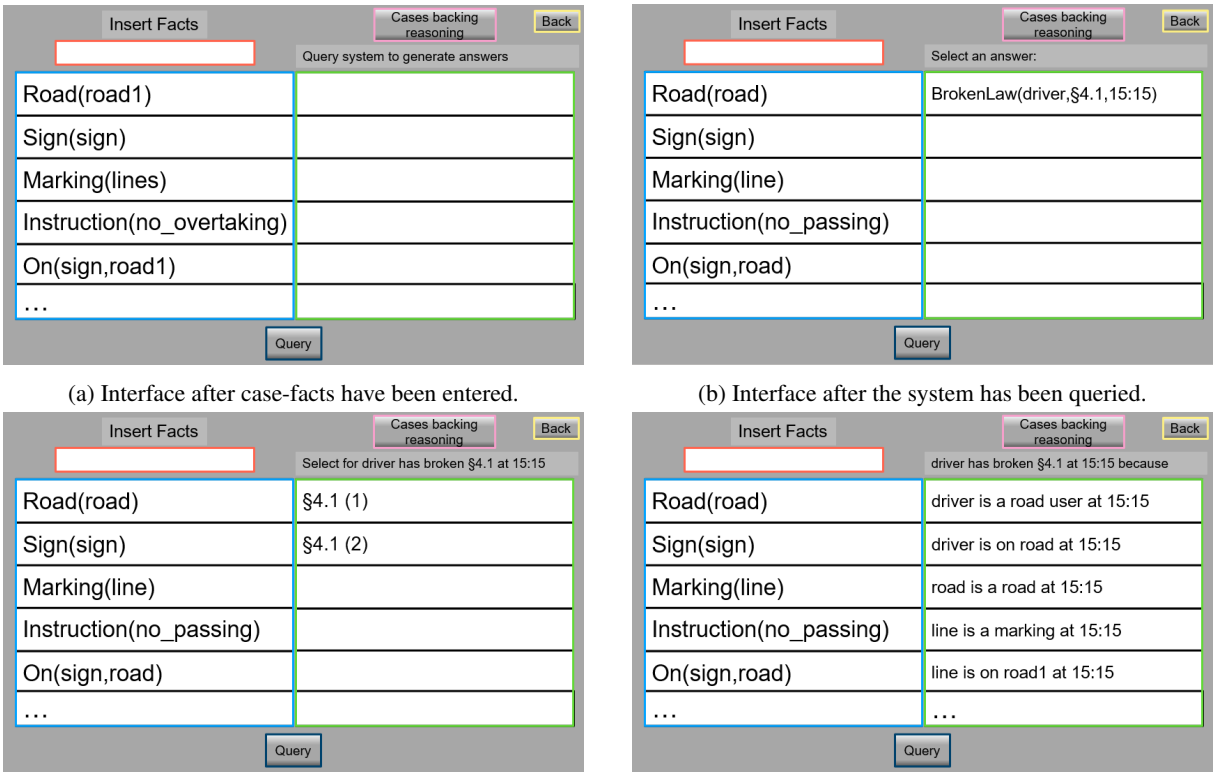


Figure 3: Explanation generated by FÆRDXEL for query `BrokenLaw(P,X,T)` on Example 3.



(a) Interface after case-facts have been entered.

(b) Interface after the system has been queried.

(c) Interface after an answer has been selected.

(d) Interface after an explanation has been selected.

Figure 4: Illustrations of the interface. The three horizontal dots at the bottom of a display box informs the user that additional items can be found by scrolling.

Figure 4a shows the interface after all facts about the case have been entered and Figure 4b after the reasoning algorithm has been run. The left side of the interface displays the facts entered for the case, and the right side shows answers and explanations to queries. All facts are entered using the red box under "Insert Facts", and must be written in Datalog using the predefined predicates. Entered facts are shown in the blue box on the left side, and can be removed again by clicking them. The reasoning algorithm runs the query `BrokenLaw(P,X,T)` on the entered facts when the dark blue "Query" button is pressed, after which the answers that have been found are shown in the green box on the right side.

Clicking on an answer selects it, and replaces the right side of the window with a list of immediate explanations (the child nodes of the root in the tree generated by the query, Figure 4c). At this point the yellow "Back" button in the top right corner becomes active, allowing the user to reverse the last selection performed in the green box.

Figure 4d shows the selected immediate explanation for the previously selected answer. The green box now displays the premises of the rule used to derive the answer. Pressing the "Cases backing reasoning" button shows references to the paragraphs (or cases, or both) from which the rule is derived. Further clicking on the immediate explanations shown will in turn exhibit their own immediate explanations (child nodes on the tree). Eventually, every branch of the tree will end with some leaf nodes, which are exactly the facts from the case (or from the knowledge base).

As shown, the interface allows for good traversability and understandability of explanations.

Case Nr.	FÆRDXEL's Conclusion	Court's Conclusion
1	§ 3	§ 3
2	None	§ 82
3	§ 3, § 15	§ 15
4	§ 3, § 15	§ 3, § 15
5	§ 42, § 44	§ 42
6	§ 3, § 15, § 54	§ 3, § 15, § 54
7	§ 42, § 53	§ 42, § 53
8	§ 56	§ 117
9	§ 3, § 15, § 54	§ 3, § 15, § 54
10	§ 3, § 9, § 15, § 53	§ 3, § 9, § 53

Table 1: Conclusions on what laws were breached according to FÆRDXEL and the courts respectively for 10 different traffic law trial. Red color indicates incorrect conclusion, and green color indicates correct conclusion but not what the defendant was on trial for.

7 Evaluation and Target Users

Two aspects of FÆRDXEL have been evaluated: the correctness of its conclusions and the value it might add to the Danish legal system. To justify the choice of these evaluations, we first describe the intended use of the tool.

FÆRDXEL is intended to be used on Danish traffic law cases, by having a user enter facts about the case, allowing the tool to providing sound arguments for and against law-breaches. Its purpose is to assist legal professionals, and provide argumentations in situations that relies on law and legal praxis that has been thoroughly decided. FÆRDXEL is not intended to be a judge of facts or of judgements – this task is performed by the court system.

Our first evaluation intends to ascertain whether FÆRDXEL is correct, in the sense that its conclusions are valid given the facts entered as input. Our second evaluation looks into the more empirical assessment of whether FÆRDXEL indeed adds value to professionals working within the Danish legal system, and identifies concrete areas of applicability.

Evaluation of correctness. The correctness of FÆRDXEL's conclusions was tested by applying the system to on-going court cases, and analysing the differences between the breaches of law deduced by FÆRDXEL and those reached by the court. The system was applied by having an expert user entering the facts of the case in the Datalog format. All facts were entered during the trial, and before the defendant was given a verdict. These findings are summarized in Table 1.

The system was tested on ten court cases. For five of the cases (1, 4, 6, 7 and 9) there were no difference between the conclusions of FÆRDXEL and the conclusions of the courts.

For two of the cases (3, 10), FÆRDXEL concluded an additional breach of law beyond those reached by the courts. In both cases, the defendant admitted to additional facts during trial that led to those conclusions, but the Danish courts are only allowed to judge on breaches brought forth prior to trial, and those additional breaches were not among them. In a real-life setting, these additional facts would not be introduced into FÆRDXEL, and these additional breaches would not be derived.

In case 5, which was about speeding on the highway, FÆRDXEL incorrectly deduced that the defen-

dant broke §44, concerning which vehicles are allowed on the highway, because the expert user forgot to enter that the vehicle the defendant drove was a car. After adding this fact, the system again derived the correct answer.

A similar situation occurs with case 8, about a driver with a revoked license. The system incorrectly deduced a breach of §56 (driving before acquiring a license) instead of §117 (driving after getting your license suspended). This was due to the expert user incorrectly inputting that the defendant did not have their license instead of having their license suspended, and fixing the input again led to the correct conclusion.

The remaining case (2) deals with a chair falling from a trailer, and the defendant was found to be innocent by FÆRDXEL. Due to a mistake, an information hidden in case law was not included in the translation of the law to rules. Therefore the system incorrectly assumed that for §82, concerning unsecured goods, a person needed to be in possible danger, when actually it only needs to be likely for a person to be where the goods fell. This is the only case where FÆRDXEL's knowledge was incorrect, but in a similar way to humans, the knowledge of the system can be updated, ensuring it will not make the same mistake again. After updating the rule, FÆRDXEL was able to derive the right conclusion.

Since FÆRDXEL provides explanations for all conclusions, it is easy to refute false positives, such as in case 5 and partly in case 8. However, since the system can not explain why it did not deduce a specific conclusion, it is harder to debunk false negatives. Specifically, detecting the error in case 2 requires knowing the relevant legislation.

In the context of using FÆRDXEL as a tool in the courts, the fact that false positives are easier to detect than false negatives is aligned with the principle in Western court systems that it is preferable to let a guilty man go free than convict an innocent one (presumption of innocence).

Evaluation of social value. To evaluate the social value of FÆRDXEL, the system has been presented to a group of Danish legal experts, with the goal of getting feedback on the system and its potential uses. Initially these experts were sceptical, but after understanding the mechanisms of the system better, they showed excitement for the project. They believe that FÆRDXEL and similar tools can provide great help in a number of areas. They also identified bottlenecks for usability, pointing out that only an expert can input facts in Datalog – an instance of a well-known general issue with expert systems [11]. They also identified four possible groups of end-users who would benefit from FÆRDXEL: Traffic police, prosecutors, citizen and judges.

Traffic police officers conduct multiple traffic stops a day, each of which has to be supported by a valid legal reason. These officers have a quick-reference guide to assist them in conducting valid stops, however as the law is quite complex, the guide is a simplification of the actual legal current praxis. This creates false positives that need to be solved in court, adding to the workload of the legal system. Replacing this quick-reference guide by a tool such as FÆRDXEL would likely help reduce the number of false positives, due to FÆRDXEL's better understanding of Danish traffic law. However, this application requires an improvement to the current interface, as any viable system would need to be as fast to use as the existing guide.

Prosecutors examine hundreds of potential cases per month and need to make quick decisions about which ones are solid enough to merit the effort of prosecution. They typically rely on assistants (very often law students) to suggest different possible arguments, whose strength they then evaluate. One known problem is that these assistants occasionally make hard-to-detect typos that effect the validity of the case. Using FÆRDXEL as an additional assistant could heavily reduce the workload of prosecutors, while also reducing the amount of minor errors.

End-User	*Benefits*	*Requirements of* FÆRDXEL
Traffic Police	Identifying law-breaking drivers	Fast and mostly automatic
Prosecutors	Providing legal arguments	Fast to use
Citizens	Answering legal questions	Usable by non-expert
Judges	Providing counter-arguments	Reason about punishment

Table 2: Summarized feedback from legal experts about possible end-user for FÆRDXEL the way they could benefit, and the improvement needed for that type of user.

Common citizens involved in a legal issue often have to analyse the cost-benefits of hiring a (likely expensive) lawyer for a specific case. This results citizens often representing themselves in smaller cases. To mitigate the disadvantages of the citizen not having a lawyer, the law requires that both the judge and prosecutor fairly defends the interests of the citizen, adding an additional work load to both judges and prosecutors. The legal system as a whole could benefit immensely from an online system that would help citizens understand why they are being accused of breaking the law and whether it is meaningful to fight the accusation in court.

Finally, *judges* need to make rulings on cases and decide on sentencing, with limited time to do so. They also rely on assistants who look for similar cases and present arguments for heavier or lighter sentencing. In this domain, fairness is the dominating criterion meaning that FÆRDXEL would unlikely be useful as a replacement of assistant – but it could still very useful as an additional source of thorough arguments.

The end-user group of judges was further explored by allowing a judge to interact with the system before and after a trial, and getting their feedback on needed improvement. The judge heavily emphasised that reasoning about punishment, would be the most useful aspect for a judge.

All findings on use-cases are summarized in Table 2.

8 Conclusions and Future Work

The entire Danish traffic law has been manually encoded into FÆRDXEL, allowing it to answer queries regarding legal cases using SLD-resolution and provide explanations behind its answers. The implementation is available at https://doi.org/10.5281/zenodo.15626892. Our empirical evaluation highlights the potential that can arise out of the collaboration between FÆRDXEL and a human user, with our panel of legal expert identifying multiple suitable application domains. We plan to explore these applications and whether we can improve the quality of the explanations provided by FÆRDXEL.

As a next step, we want to extend FÆRDXEL's reasoning capabilities to the domain of sentencing. Danish traffic law states a range of punishment for a given crime and defines aggravating and mitigating circumstances, but it is up to the judge to decide what the exact fine or prison sentence should be. We plan to explore how to apply of Fuzzy Logic to this task, by viewing 'aggravating' and 'mitigating' as Fuzzy terms in the domain of possible punishments. A similar approach, from which we draw inspiration, has been applied to Polish punishment law [18].

Lastly, to improve the usability of FÆRDXEL, we are exploring how to use large language models in a reliable way to convert written natural language about a case into the Datalog format that the tool requires as input. We believe that this could be the way to develop a user-friendly interface that could

make FÆRDXEL easily accessible to non-experts. The major challenge is to ensure that the translation is correct, requiring the user to confirm that the model's "understanding" of the input is the right one.

References

[1] Alaa N. Akkila & Samy S. Abu Naser (2016): *Proposed Expert System for Calculating Inheritance in Islam*. *World Wide Journal of Multidisciplinary Research and Development* 2(9), pp. 38–48.

[2] Adrien Bibal, Michael Lognoul, Alexandre de Streel & Benoît Frénay (2020): *Legal requirements on explainability in machine learning*. *Artifcial Intelligence and Law* 29, pp. 149–169, doi:10.1007/s10506-020-09270-4.

[3] Zhuo Chen (2016): *Automating Disease Management Using Answer Set Programming*. In Manuel Carro, Andy King, Neda Saeedloei & Marina De Vos, editors: *Technical communications of ICLP*, *OASIcs* 52, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 22:1–22:10, doi:10.4230/OASICS.ICLP.2016.22.

[4] Andrew Coan & Harry Surden (2025): *Artificial Intelligence and Constitutional Interpretation*. *University of Colorado Law Review* 413, doi:10.2139/ssrn.5018779.

[5] Counsel AI Corporation (2025): *Harvey AI*. Available at https://www.harvey.ai/.

[6] Danmarks Domstole (2022): *Domsdatabasen*. https://domsdatabasen.dk/#sag/2526/3008. [Accessed 23-09-2024].

[7] Basel Y. Elhabil & Samy S. Abu-Naser (2021): *An Expert System for Ankle Problems*. *International Journal of Engineering and Information Systems (IJEAIS)* 5(4), pp. 57–66.

[8] Mohammad Farajollahi & Vahid Baradaran (2024): *Expert System Application in law: A review of research and applications*. *International Journal of Nonlinear Analysis and Applications* 15(8), pp. 107–114, doi:10.22075/ijnaa.2023.31260.4596.

[9] Herve Gallaire, Jack Minker & Jean-Marie Nicolas (1984): *Logic and Databases: A Deductive Approach*. *ACM Comput. Surv.*, p. 153–185, doi:10.1145/356924.356929.

[10] Ryan Greenblatt, Carson Denison, Benjamin Wright, Fabien Roger, Monte MacDiarmid, Samuel Marks, Johannes Treutlein, Tim Belonax, Jack Chen, David Duvenaud, Akbir Khan, Julian Michael, Sören Mindermann, Ethan Perez, Linda Petrini, Jonathan Uesato, Jared Kaplan, Buck Shlegeris, Samuel R. Bowman & Evan Hubinger (2024): *Alignment faking in large language models*. *CoRR*, doi:10.48550/arXiv.2412.14093.

[11] Graham Greenleaf, Andrew Mowbray & Philip Chung (2018): *Building sustainable free legal advisory systems: Experiences from the history of AI & law*. *Comput. Law Secur. Rev.* 34(2), pp. 314–326, doi:10.1016/J.CLSR.2018.02.007.

[12] Mohammad Shahadat Hossain, Mohammad A. Haque, Rashed Mustafa, Razuan Karim, Hirak Chandra Dey & Md. Yusuf (2016): *An Expert System to Assist the Diagnosis of Ischemic Heart Disease*. *International Journal of Integrated Care* 16, p. 31, doi:10.5334/ijic.2974.

[13] Inc Ironclad (2025): *Ironclad*. Available at https://ironcladapp.com/product/ironclad-for-legal/.

[14] Somay Jain, Mukul Aggarwal, Yash Singhal & Apri Dwi Lestari (2023): *An expert system on diagnosis of mental diseases*. *Journal of Soft Computing Exploration* 4, pp. 53–58, doi:10.52465/joscex.v4i1.100.

[15] LexisNexis (2025): *Lexis+AI*. Available at https://www.lexisnexis.com/en-us/products/lexis-plus-ai.page.

[16] Yang Liu, Yuanshun Yao, Jean-Francois Ton, Xiaoying Zhang, Ruocheng Guo, Hao Cheng, Yegor Klochkov, Muhammad Faaiz Taufiq & Hang Li (2024): *Trustworthy LLMs: a Survey and Guideline for Evaluating Large Language Models' Alignment*. *arXiv preprint*, doi:10.48550/arXiv.2308.05374.

[17] John W. Lloyd (1984): *Foundations of Logic Programming*. Springer.

[18] Michal Lower & Monika Lower (2019): *Fuzzy Inference Model for Punishing a Perpetrator in a Judicial Process*. In Ngoc Thanh Nguyen, Richard Chbeir, Ernesto Exposito, Philippe Aniorté & Bogdan Trawiński, editors: *Procs. ICCCI19*, Springer, pp. 463–473, doi:10.1007/978-3-030-28377-3_38.

[19] Genie AI Ltd. (2025): *Genie AI*. Available at https://www.genieai.co/en-us.

[20] L. Thorne McCarty (1977): *Reflections on TAXMAN: An Experiment in Artificial Intelligence and Legal Reasoning*. Harvard Law Review 90, pp. 837–893.

[21] Thomas A. O'Callaghan, James Popple & Eric McCreath (2003): *SHYSTER-MYCIN: a hybrid legal expert system*. In: *Procs. ICAIL '03*, Association for Computing Machinery, p. 103–104, doi:10.1145/1047788.1047814.

[22] Kamalendu Pal & John A. Campbell (1997): *An application of rule-based and case-based reasoning within a single legal knowledge-based system*. SIGMIS Database 28(4), p. 48–63, doi:10.1145/277339.277344.

[23] James Popple (1993): *SHYSTER: A Pragmatic Legal Expert System: PhD thesis*. Australian National Univ.

[24] Thomson Reuters (2025): *CoCounsel*. Available at https://www.thomsonreuters.com/en.

[25] Alessandro Ronca, Mark Kaminski, Bernardo Cuenca Grau, Boris Motik & Ian Horrocks (2018): *Stream Reasoning in Temporal Datalog*. In Sheila A. McIlraith & Kilian Q. Weinberger, editors: *Procs. AAAI*, AAAI Press, pp. 1941–1948, doi:10.1609/aaai.v32i1.11537.

[26] Kathryn E. Sanders (1991): *Representing and reasoning about open-textured predicates*. In: *Procs. ICAIL*, Association for Computing Machinery, p. 137–144, doi:10.1145/112646.112663.

[27] Daniel Schwarcz & Jonathan H. Choi (2023): *AI Tools for Lawyers: A Practical Guide*. Minnesota Law Review Headnotes 1, doi:10.2139/ssrn.4404017.

[28] Henrik Waaben & Kirsten Søndergaard Munck (2023): *Færdselsloven med kommentarer*. Djøf.

[29] Yixin Wan, George Pu, Jiao Sun, Aparna Garimella, Kai-Wei Chang & Nanyun Peng (2023): *"Kelly is a Warm Person, Joseph is a Role Model": Gender Biases in LLM-Generated Reference Letters*. In Houda Bouamor, Juan Pino & Kalika Bali, editors: *Procs. EMNLP*, Association for Computational Linguistics, pp. 3730–3748, doi:10.18653/v1/2023.findings-emnlp.243.

[30] Chengen Wang & Murat Kantarcioglu (2025): *A Review of DeepSeek Models' Key Innovative Techniques*. arXiv preprint, doi:10.48550/arXiv.2503.11486.

[31] Yufei Wang, Wanjun Zhong, Liangyou Li, Fei Mi, Xingshan Zeng, Wenyong Huang, Lifeng Shang, Xin Jiang & Qun Liu (2023): *Aligning Large Language Models with Human: A Survey*. arXiv preprint, doi:10.48550/arXiv.2307.12966.

[32] J Zeleznikow & A Stranieri (1998): *Split up: an intelligent decision support system which provides advice upon property division following divorce*. International Journal of Law and Information Technology 6(2), pp. 190–213, doi:10.1093/ijlit/6.2.190.

Explainable AI for Sperm Morphology: Integrating YOLO with FastLAS

Talissa Dreossi Agostino Dovier Francesca Corte Pause Susy Urli Martina Crociati
Università degli Studi di Udine, Udine, Italy
dreossi.talissa@spes.uniud.it
{agostino.dovier,francesca.cortepause,susy.urli,martina.crociati}@uniud.it

Introduction

Deep learning models, although capable of achieving exceptional accuracy, are often regarded as "black boxes" due to their opaque internal decision-making processes. This opacity is particularly problematic in fields like medical diagnostics, where understanding the reasoning behind a classification is as important as the classification itself. *Logic Programming* (LP), with its inherent interpretability, offers a promising complementary approach. To reconcile this trade-off between accuracy and transparency, this study proposes a hybrid AI system that combines deep learning and symbolic reasoning. The focus of this work is the morphological classification of bull spermatozoa, which is a critical step in breeding selection, that traditionally depends on labour-intensive visual inspection by trained operators and suffers from inter-operator variability. Automating this task offers the dual benefit of efficiency and consistency.

The widespread adoption of AI systems in sensitive domains, such as medicine and veterinary science, has led to notable improvements in diagnostic accuracy and operational efficiency [23, 2]. Automated tools like Computer-Aided Sperm Analysis (CASA) are already employed to evaluate sperm viability and motility [19], while AI models have shown predictive capabilities in applications ranging from calving time estimation [7, 3] to detecting lung abnormalities in cats using deep learning [13]. However, the increasing reliance on AI in these high-stakes contexts has heightened the need for explainability to foster trust and support informed decision-making.

Our solution, eXplainable AI for Bull Sperm Classification (XAI-BSC), integrates the *You Only Look Once* (YOLO) [22, 16] object detection algorithm for spermatozoa identification and classification with *Inductive Logic Programming* (ILP) for rule-based explanation. Specifically, YOLO is employed to localise and categorise sperm cells in microscopic images, while FastLAS [17], an ILP framework, used to induce symbolic classification rules on YOLO's predictions. These rules are encoded as an *Answer Set Programming* (ASP) program [14]. To facilitate explanation, the system leverages xASP2 [1, 25], a framework that links the derivation of atoms in answer sets to their underlying logical justifications. Unlike related systems such as xclingo [4], DiscASP [18], or exp(ASPc) [24], xASP2 supports the explanation of false atoms and provides richer language features.

YOLO-based object detection has already shown efficacy in various medical imaging tasks, including cancer detection and cell identification [5, 21], while FastLAS has demonstrated strong performance in domains requiring interpretable models, such as security policy enforcement [12], weather prediction [11], and legal reasoning [9]. Its integration with neural models [8, 6] further underscores the feasibility of combining data-driven learning with symbolic explainability.

While the current XAI-BSC prototype requires limited human input during the feature extraction phase, it already delivers interpretable, rule-based explanations in natural language, significantly re-

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 397–404, doi:10.4204/EPTCS.439.27

© Dreossi, Dovier, Corte Pause, Urli & Crociati
This work is licensed under the
Creative Commons Attribution License.

ducing the time required for expert analysis and supporting reproducible, justifiable decisions in sperm morphology assessment.

Methods

Accurate assessment of bull spermatozoa morphology is crucial for evaluating fertility, traditionally performed via visual evaluation under bright-field microscopy following *Society For Theriogenology* (SFT) guidelines. This process typically involves at least two technicians, with agreement ranging from 43% (Fair) to 58% (Moderate), ensuring reliability. Our study employs two key techniques for this evaluation: YOLOv8 for object detection and FastLAS for learning interpretable rules.

Background

YOLOv8 is an efficient one-stage object detection model that performs localization and classification in a single pass. Unlike two-stage models, YOLOv8 processes images through a *Convolutional Neural Network* (CNN) to extract features. It divides the image into an $S \times S$ grid, with each cell predicting bounding boxes and class probabilities. YOLOv8 uses an anchor-free detection approach, speeding up computation by avoiding predefined anchor boxes. Its architecture comprises a backbone (pre-trained CNN for feature extraction), a neck (Feature Pyramid Network for combining features and detecting varying object sizes), and a head (computing bounding box coordinates (x, y, w, h), confidence scores, and class probabilities).

With ILP, we are able to learn logical rules from examples and background knowledge. We chose an ILP framework over expert-defined rules due to the difficulty in capturing subtle morphological features with fixed thresholds. The process involves searching an hypothesis space for rules that best explain given examples while being consistent with background knowledge. We adopted *Learning from Answer Sets* (LAS), which leverages ASP for learning non-monotonic logical programs. A LAS learning problem is defined as $T = \langle B, S_M, E \rangle$, where B is background knowledge (ASP program), S_M is the hypothesis space (rules to be learned), and E is the set of examples the hypothesis must explain. The hypothesis space is implicitly defined by mode bias $\langle M_h, M_b \rangle$, specifying allowed predicates and argument types for rule heads and bodies. Examples are instead partial interpretations $\langle e^{inc}, e^{exc} \rangle$, where e^{inc} are inclusions (atoms that must be true) and e^{exc} are exclusions (atoms that must be false). LAS also handles context-dependent partial interpretations (CDPIs) $e_{cdpi} = \langle e_{pi}, e_{ctx} \rangle$, where e_{ctx} is an ASP program representing the context. A program P accepts e_{cdpi} if a stable model A of $P \cup e_{ctx}$ extends e_{pi}. FastLAS [17] excels at handling noisy data by associating a penalty e_{pen} with each example, allowing some violations if it leads to a better overall hypothesis. A Noisy LAS task $T^{noise} = \langle B, S_M, E \rangle$ aims to find an optimal hypothesis $H \subseteq S_M$ that minimizes a cost function (sum of penalties for unaccepted examples plus hypothesis length).

Regarding explainability we have xASP2 [1] which is a tool designed to explain computed answer sets, based on justification graphs [20]. Given a program P_i, an answer set S, and an atom α, xASP2 answers "Why does $\alpha \in S$ (resp., $\alpha \notin S$)?" by identifying the supporting subset of P_i. It generates explanations as justification graphs (*Directed Acyclic Graphs*, DAGs) where nodes are atoms and edges are rules. Edges from A to B mean B is in the rule's body and A in its head. For each atom in a stable model, xASP2 finds a supporting rule whose body is true and contains only already-explained atoms, ensuring acyclicity. The explanation graph visualizes atom relationships, tracking rules used at each derivation step. xASP2 uses a metaprogramming approach with the Clingo Python API, offering a

flexible interface for requesting explanations for literals or generating multiple graphs, with an interactive web-based environment for visualization.

The Model

To train YOLO, the initial dataset that we used comprised 4,890 microscopy fields containing 1 to 20 eosin-nigrosine-stained spermatozoa, expertly labeled. Data augmentation (brightness,contrast, blur, noise) expanded the dataset to 8,243 images, annotated with bounding boxes using labelImg [26]. Spermatozoa were classified into six categories: *Normal Alive* (NA), *Normal Dead* (ND), *Major Abnormalities Alive* (MAA), *Major Abnormalities Dead* (MAD), *Minor Abnormalities Alive* (mAA), and *Minor Abnormalities Dead* (mAD). Training was set to 100 epochs with early stopping, extending by 50 epochs if performance improved. The optimal model was saved based on validation loss, typically around 100 epochs. The YOLO network was used without customization for this initial work. More information can be found in our previous works [28, 27].

The model [27] achieved 82% mean accuracy, though performance varied due to class imbalance. Prediction quality was assessed using the *Intersection over Union* (IoU) metric: $IoU = \frac{GT \cap Pr}{GT \cup Pr}$, where GT and Pr are the areas of ground-truth and predicted boxes, respectively. mAP50 (*mean Average Precision*, IoU ≥ 0.50) reached 79%, and mAP50-95 (IoU 0.50-0.95) was 44%, reflecting detection and localization performance. An 82% accuracy is considered satisfactory, as an 80-85% agreement coefficient between two operators is deemed "moderate to very strong" in real-world scenarios [15].

On the other hand, as background knowledge for the ILP module we just encoded predicates for the domain of the morphological features. To extract those features, and then use them, we used XAI-BSC. To ensure the ILP module's independence from black-box techniques, morphological information retrieval was achieved through traditional computer vision. XAI-BSC takes YOLO annotation files and corresponding images as input, displaying one spermatozoon at a time and prompting the user for morphological characteristics. It then attempts to outline the sperm head contour, crucial for classification (e.g., pear-shaped heads indicate major anomalies). Head contour detection involves a two-step process: first, *Hough Circle Transform* (HCT) identifies candidate head regions based on bright, approximately circular appearance, using the Canny edge detector and accumulator space voting. Second, active contour models (snakes) refine detection by accurately tracing contours. A snake is initialized as a larger circle around the detected center and evolved over a Gaussian-smoothed image to converge on the actual boundary. However, low contrast in images and multiple sperm heads can make automatic contour detection challenging. When contours are not identifiable (Figure 1, left), the app defaults to bounding box data, measuring redness (for viability), ratio (for tail morphology), and area (for size). If contours are correctly detected (Figure 1, right), additional features like contour length, enclosed area, ratio of minimum/maximum axes, and roundness ($Roundness = \frac{4\pi A}{p^2}$, where A is the area and p is the perimeter) are extracted. Tail detection is even more challenging due to varying shapes, but, for example, bounding box aspect ratio provides some insight (closer to one indicates curled, coiled, or missing tail).

Two separate training sessions were conducted: one for distinguishing alive from dead spermatozoa, and another for differentiating between no anomalies ("n"), major anomalies ("M"), and minor anomalies ("m"). This modular approach facilitated ILP learning. Examples are encoded as `#pos(id@w, {inc}, {exc}, {cntx})`, where `#pos` indicates a positive example, `id` is a unique identifier, `w` is the weight (representativeness), `inc` are inclusions (atoms that must be true), `exc` are exclusions (atoms that cannot be true), and `cntx` is the context. For instance:

```
#pos(id58_12@100,
    {label("M")},                                   % inclusions
```

Figure 1: Examples of contour detection of the head. Correct: green/right, Wrong: any of them/right.

```
{label("n"), label("m"), curledTail,
missingTail, singleCoilTail, overtunedHead},    % exclusions
{red(648). area (25200). ratio (175).
headRoundness (74). headArea (2594).
lenghtHead (210). ratioHead(2). bubbleHead.}).  % context
```

All examples were assigned equal weight (100) due to balanced class numbers and lack of specific insight into representativeness or noise. To improve generalizability and address ILP scalability, 10-fold cross-validation was used, with each training fold containing 12 spermatozoa per class and test sets containing 7 per class, reducing overfitting and ensuring robustness.

FastLAS uses a penalty function to guide rule learning by assigning costs to rule components, prioritizing relevant features. For distinguishing alive from dead spermatozoa, a strong bias is introduced: a penalty of 100 for the rule head and 1 for each body predicate. This forces rules to cover multiple examples, as failing to cover one incurs a significant cost of 100. For classifying normal, major, or minor anomalies, a different penalty scheme is used. The rule head penalty is lower (10), reflecting the wider range of morphological features determining anomalies. Body predicates are categorized: features specific to a class of anomalies (e.g., missing or curled tail) have the lowest cost (3), as they are strong indicators. Other features have slightly higher costs: redness (5) is more relevant for viability, and head length is less informative. Head area, ratio, and roundness have intermediate penalties (4). These penalty differences, though small, balance rule complexity and generalizability, ensuring learned rules prioritize discriminative characteristics while avoiding overfitting.

Results

As expected, FastLAS exhibits lower predictive performance compared to YOLO due to their fundamentally different natures; however, its primary goal is explainability, not outperforming deep learning. FastLAS rules, such as

```
pred_label("m"):- distalDroplet, ratio(V_0_rr), V_0_rr >= 77.
pred_label("alive"):- red(V_0_r), V_0_r <= 386.
```

align with experts' knowledge. When compared against traditional machine learning models (SVM, Decision Trees, Random Forests), FastLAS consistently outperforms them across various metrics, demonstrating its ability to extract meaningful rules from data (Figure 2).

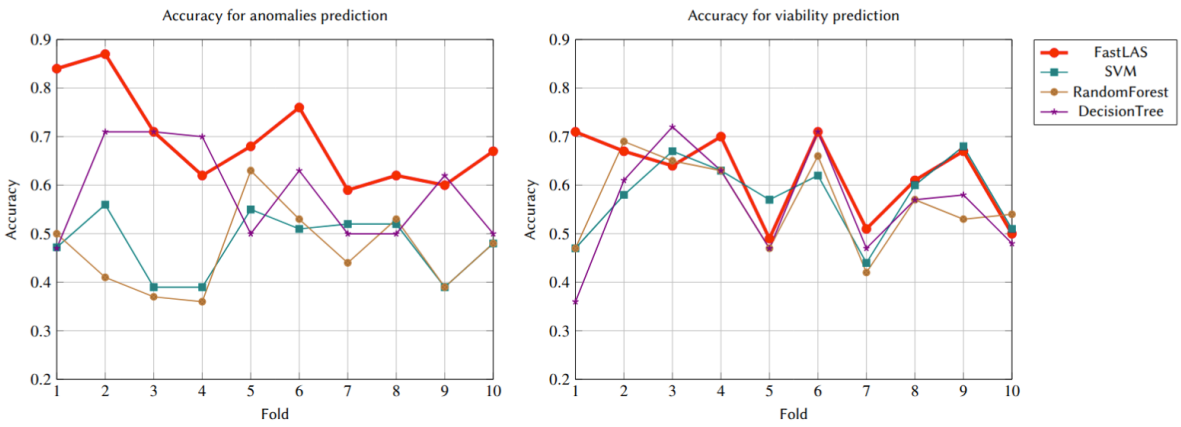


Figure 2: Accuracy for each fold for each model. On the left, the graph relative to prediction of anomalies and, on the right, the one relative to viability.

Overall, while FastLAS's performance is lower than YOLO's, it remains competitive with traditional ML models and offers the crucial advantage of explainability. It is important to note that YOLO is used for prediction (with over 80% accuracy), and FastLAS rules are used for explainability, as YOLO was trained on a wider dataset with a different preprocessing process.

Explainability

The ASP rules we obtained from FastLAS are extremely interpretable, but we still need to reconstruct the reasoning process, i.e. the rules that are triggered to get the correct prediction. For this purpose, we used xASP2 in XAI-BSC app. The workflow begins when a user uploads an image, which YOLO processes to detect and classify spermatozoa. The model's predictions are saved, and the image, overlaid with bounding boxes and predicted labels, is presented. The user can then select a specific spermatozoon for which they desire an explanation. Upon selection, the chosen spermatozoon is displayed in isolation using the sub-image defined by its predicted bounding box. Similar to the FastLAS dataset preparation, XAI-BSC collects some morphological information directly from the user, while other features are extracted automatically. These features are then encoded into logical facts in ASP format. The feature encoding and the rules learned by FastLAS are fed as input to Clingo, which computes the answer set corresponding to the classification process. Using xASP2, a graph representing this answer set is generated. It is crucial to note that ASP rules may occasionally yield no prediction or incorrect predictions: to ensure explanations align with YOLO's output, an auxiliary predicate (`found_correct_explanation`) is included, which holds true only when the ASP-derived label matches YOLO's prediction; other ASP predictions are ignored. Once the correct reasoning path is identified from the xASP2 DAG, it is translated into natural language, providing a clear and understandable explanation for the classification decision. The algorithm traces links from the correct explanation node to feature or fact nodes, extracting variable values for substitution into explanations. XAI-BSC and tests were conducted on a system running Ubuntu 24.04 LTS with an Intel 12th Gen Core i7-12700 processor (max 4.9 GHz), 16 GB RAM, an NVIDIA GeForce RTX 3060 GPU, and an SK Hynix NVMe SSD.

Figure 3 (left/center) illustrates an explanation for a sample predicted as having a major anomaly. The system identifies the presence of a proximal droplet as the key justifying feature, presented both

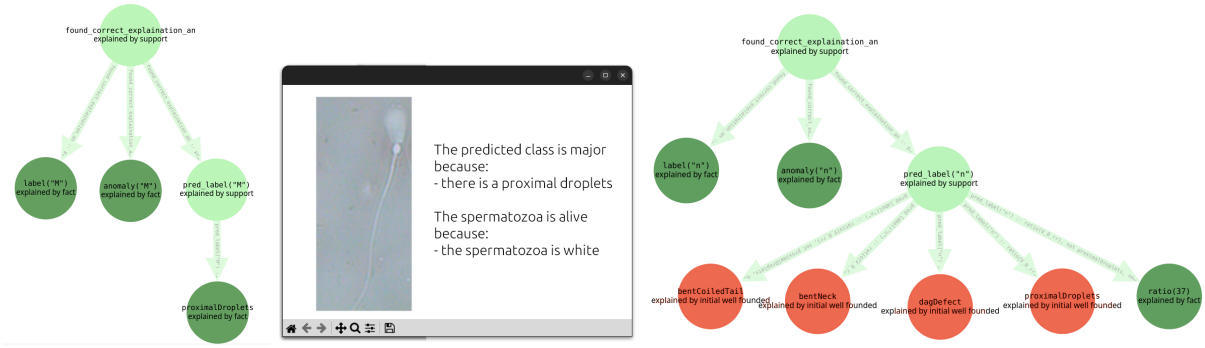


Figure 3: xASP2 DAG for the anomaly prediction (left) with explanation in XAI-BSC app (center), and xASP2 DAG for a prediction of no anomalies (right).

in natural language alongside the spermatozoon image and as an explanation graph. Figure 3 (right) shows the explanation for a sample classified as having no anomalies. In this instance, the classification is supported by the absence (indicated by red nodes) of specific features like bent tail, bent neck, or proximal droplets. Additionally, a ratio value of 0.37 (scaled for ASP encoding) further supports the prediction of a normal spermatozoon.

Conclusions and Future Works

By integrating YOLO for object detection with FastLAS for rule learning, we have successfully developed XAI-BSC, a tool that achieves both accurate classification and interpretable explanations for bull spermatozoa morphology. While FastLAS presented scalability limitations with increasing features, examples, or hypothesis space, we mitigated this by restricting learning to separate sub-tasks (viability and anomaly classification). To enhance generalizability and robustness, we employed a 10-fold cross-validation strategy, ensuring rules were learned from a diverse set of examples and were not overfitted to specific dataset splits.

Looking ahead, several improvements and extensions are planned. Our immediate goals include enhancing the accuracy of sperm head detection and developing reliable methods for tail detection and automating the identification of other morphological features, such as proximal droplets. These advancements will significantly reduce manual input and increase the system's autonomy. Furthermore, we intend to conduct comprehensive user studies to evaluate the practical effectiveness of XAI-BSC's explainability features, assessing how well it supports decision-making and fosters user trust in real-world scenarios. Finally, we plan to compare our hybrid approach with other explainable AI methods, such as LIME and SHAP, to further benchmark its performance and interpretability benefits. Further details on the content of the research summarized in this report can be found in [10].

Acknowledgments. The authors extend their gratitude to Prof. Alessandro Dal Palù for his invaluable support in the elaboration of image with computer vision techniques and Prof. Giuseppe Stradaioli for his assistance during the early stages of the YOLO-based application. The work is partially supported by Interdepartment Project on AI (Strategic Plan of UniUD 2022-25).

References

[1] M. Alviano, L. L. Trieu, T. C. Son & M. Balduccini (2023): *Explanations for answer set programming*. In: *Proceedings of the 39th International Conference on Logic Programming (ICLP 2023)*, *Electronic Proceedings in Theoretical Computer Science* 385, London, UK, pp. 27–40, doi:10.4204/eptcs.385.4.

[2] R. B. Appleby & P. S. Basran (2022): *Artificial intelligence in veterinary medicine*. *Journal of the American Veterinary Medical Association* 260, pp. 819–824, doi:10.2460/javma.22.03.0093.

[3] M. Borchers, Y. Chang, K. Proudfoot, B. Wadsworth, A. Stone & J. Bewley (2017): *Machine-learning-based calving prediction from activity, lying, and ruminating behaviors in dairy cattle*. *Journal of Dairy Science* 100, pp. 5664–5674, doi:10.3168/jds.2016-11526.

[4] P. Cabalar, J. Fandinno & B. Muniz (2020): *A system for explainable answer set programming*. In: *Proceedings of the 36th International Conference on Logic Programming (ICLP 2020)*, *Electronic Proceedings in Theoretical Computer Science* 325, Rende, Italy, pp. 124–136, doi:10.4204/eptcs.325.19.

[5] X. Chen, H. Zheng, H. Tang & F. Li (2024): *Multi-scale perceptual YOLO for automatic detection of clue cells and trichomonas in fluorescence microscopic images*. *Computers in Biology and Medicine* 175, p. 108500, doi:10.1016/j.compbiomed.2024.108500.

[6] M. Collevati, T. Eiter & N. Higuera (2024): *Leveraging neurosymbolic AI for slice discovery*. In: *18th International Conference on Neural Symbolic Learning and Reasoning*, Springer Nature Switzerland, pp. 403–418, doi:10.1007/978-3-031-71167-1_22.

[7] M. Crociati, L. Sylla, A. De Vincenzi, G. Stradaioli & M. Monaci (2022): *How to predict parturition in cattle? a literature review of automatic devices and technologies for remote monitoring and calving prediction*. *Animals* 12, p. 405, doi:10.3390/ani12030405.

[8] D. Cunnington, A. Russo, M. Law, J. Lobo & L. Kaplan (2020): *NSL: Hybrid interpretable learning from noisy raw data*. doi:10.48550/ARXIV.2012.05023.

[9] A. Dovier, T. Dreossi & A. Formisano (2024): *XAI-LAW towards a logic programming tool for taking and explaining legal decisions*. In: *Proceedings of the 39th Italian Conference on Computational Logic (CILC 2024)*, *CEUR Workshop Proceedings* 3733, CEUR-WS.org, Rome, Italy, doi:11390/1289224.

[10] T. Dreossi, A. Dovier, F. Corte Pause, S. Urli & M. Crociati (2025): *Explainable AI for Sperm Morphology: Integrating YOLO with FastLAS*. In: *Proceedings of the 40th Italian Conference on Computational Logic (CILC 2025)*, CEUR Workshop Proceedings, CEUR-WS.org, Alghero, Italy. To appear.

[11] T. Dreossi, A. Dovier, A. Formisano, M. Law, A. Manzato, A. Russo & M. Tait (2024): *Towards explainable weather forecasting through FastLAS*. In: *Logic Programming and Nonmonotonic Reasoning*, Springer Nature Switzerland, Dallas, TX, USA, pp. 262–275, doi:10.1007/978-3-031-74209-5_20.

[12] A. Drozdov, M. Law, J. Lobo, A. Russo & M. W. Don (2021): *Online symbolic learning of policies for explainable security*. In: *Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA 2021)*, IEEE, Atlanta, GA, USA, pp. 269–278, doi:10.1109/tpsisa52974.2021.00030.

[13] L. Dumortier, F. Guépin, M.-L. Delignette-Muller, C. Boulocher & T. Grenier (2022): *Deep learning in veterinary medicine, an approach based on CNN to detect pulmonary abnormalities from lateral thoracic radiographs in cats*. *Scientific Reports* 12, doi:10.1038/s41598-022-14993-2.

[14] M. Gelfond & V. Lifschitz (1988): *The stable model semantics for logic programming*. In R. A. Kowalski & K. A. Bowen, editors: *Proceedings of the 5th International Conference and Symposium on Logic Programming*, MIT Press, Seattle, WA, USA, pp. 1070–1080. Available at `http://www.cs.utexas.edu/users/ai-lab?gel88`.

[15] R. Hanson, S. Reddick, S. Thuerauf, K. Webb, V. Kasimanickam & R. Kasimanickam (2023): *Comparison of bull sperm morphology evaluation methods under field conditions*. *Clinical Theriogenology* 15, p. 9425, doi:10.58292/ct.v15.9425.

[16] M. Hussain (2023): *YOLO-v1 to YOLO-v8, the rise of YOLO and its complementary nature toward digital manufacturing and industrial defect detection*. Machines 11, p. 677, doi:10.3390/machines11070677.

[17] M. Law, A. Russo, E. Bertino, K. Broda & J. Lobo (2020): *FastLAS: Scalable inductive logic programming incorporating domain-specific optimisation criteria*. In: 34th AAAI Conference on Artificial Intelligence (AAAI 2020), AAAI Press, New York, NY, USA, pp. 2877–2885, doi:10.1609/aaai.v34i03.5678.

[18] F. Li, H. Wang, K. Basu, E. Salazar & G. Gupta (2021): *DiscASP: A graph-based ASP system for finding relevant consistent concepts with applications to conversational socialbots*. In: Proceedings of the 37th International Conference on Logic Programming (ICLP 2021), Electronic Proceedings in Theoretical Computer Science 345, Porto, Portugal, pp. 205–218, doi:10.4204/eptcs.345.35.

[19] C. O'Meara, E. Henrotte, K. Kupisiewicz, C. Latour, M. Broekhuijse, A. Camus, L. Gavin-Plagne & E. Sellem (2022): *The effect of adjusting settings within a Computer-Assisted Sperm Analysis (CASA) system on bovine sperm motility and morphology results*. Animal Reproduction 19, doi:10.1590/1984-3143-ar2021-0077.

[20] E. Pontelli & T. C. Son (2006): *Justifications for Logic Programs Under Answer Set Semantics*. In: Proceedings of the 22nd International Conference on Logic Programming (ICLP 2006), Springer Nature Switzerland, Seattle, WA, USA, pp. 196–210, doi:10.1007/11799573_16.

[21] F. Prinzi, M. Insalaco, A. Orlando, S. Gaglio & S. Vitabile (2023): *A YOLO-based model for breast cancer detection in mammograms*. Cognitive Computation 16, pp. 107–120, doi:10.1007/s12559-023-10189-6.

[22] J. Redmon, S. Divvala, R. Girshick & A. Farhadi (2015): *You Only Look Once: Unified, real-time object detection*. doi:10.48550/ARXIV.1506.02640.

[23] M. Shehab, L. Abualigah, Q. Shambour, M. A. Abu-Hashem, M. K. Y. Shambour, A. I. Alsalibi & A. H. Gandomi (2022): *Machine learning in medical applications: A review of state-of-the-art methods*. Computers in Biology and Medicine 145, p. 105458, doi:10.1016/j.compbiomed.2022.105458.

[24] L. L. Trieu, T. C. Son & M. Balduccini (2021): *exp(ASPc): Explaining ASP programs with choice atoms and constraint rules*. In: Proceedings of the 37th International Conference on Logic Programming (ICLP 2021), Electronic Proceedings in Theoretical Computer Science 345, Porto, Portugal, pp. 155–161, doi:10.4204/eptcs.345.28.

[25] L. L. Trieu, T. C. Son & M. Balduccini (2022): *xASP: An explanation generation system for answer set programming*. In: Logic Programming and Nonmonotonic Reasoning, Springer Nature Switzerland, Genova, Italy, pp. 363–369, doi:10.1007/978-3-031-15707-3_28.

[26] D. Tzutalin (2015): *labelImg*. Available at `https://github.com/tzutalin/labelImg`.

[27] S. Urli, F. Corte Pause, T. Dreossi, M. Crociati & G. Stradaioli (2025): *Evaluation of an artificial intelligence system for bull sperm morphology evaluation*. Theriogenology 245, p. 117504, doi:10.1016/j.theriogenology.2025.117504.

[28] S. Urli, F. Corte Pause, T. Dreossi & G. Stradaioli (2024): *Implementation of an artificial intelligence system for morphological evaluation of spermatozoa in routine bull semen analysis*. In: 75th EAAP Annual Meeting, Florence, Italy. Available at `https://eaap.org/event/75th-eaap-annual-meeting-florence-italy/`.

XAI-LAW: A Logic Programming Tool for Modeling, Explaining, and Learning Legal Decisions *

Agostino Dovier Talissa Dreossi Andrea Formisano Benedetta Strizzolo
DMIF – University of Udine, Udine, Italy
Gruppo Nazionale per il Calcolo Scientifico – INdAM, Roma, Italy
agostino.dovier@uniud.it, talissa.dreossi@uniud.it
andrea.formisano@uniud.it, strizzolo.benedetta@spes.uniud.it

We propose an approach to model articles of the Italian Criminal Code (ICC), using Answer Set Programming (ASP), and to semi-automatically learn legal rules from examples based on prior judicial decisions. The developed tool is intended to support legal experts during the criminal trial phase by providing reasoning and possible legal outcomes. The methodology involves analyzing and encoding articles of the ICC in ASP, including "crimes against the person" and property offenses. The resulting model is validated on a set of previous verdicts and refined as necessary. During the encoding process, contradictions may arise; these are properly handled by the system, which also generates possible decisions for new cases and provides explanations through a tool that leverages the "supportedness" of stable models. The automatic explainability offered by the tool can also be used to clarify the logic behind judicial decisions, making the decision-making process more interpretable. Furthermore, the tool integrates an inductive logic programming system for ASP, which is employed to generalize legal rules from case examples.

1 Introduction

The desire of delegating legal decisions to an automated formal system can be traced back (at least) to Leibniz's dream. In spite of Gödel incompleteness results, holding in a more general context, several efforts have been posed in this direction since the birth of artificial intelligence, earlier expert systems, and logic programming. From the 1950s, Allen and Saxon [1, 2] paved the way for the application of logic programming in the legal field, by interpreting legal documents, and by outlining what is necessary for encoding legal rules in logic. Two of the main challenges in this area concern the inherent ambiguity of *natural language* (NL) and the applicability of legal rules.

The first notable attempt to encode laws in logic was undertaken by Kowalski et al. [35, 24, 25], who formalised part of the British Nationality Act. They introduced a specialised language, called Logical English, which consists of a *controlled natural language* (CNL), i.e. a language that easily allows English sentences to be translated into logic programs such as Prolog, Datalog, or Answer Set Programming (ASP) [25]. In this field, progress has been made through the development of systems that automatically translate NL into CNL and then into ASP code [8]. Similarly, Large Language Models (LLMs) can be employed to directly translate NL into ASP [22, 10]. A further step was to combine LLM-based semantic parsers with a learning module, such as in the aforementioned paper by Kareem et al. [22]. In that study, ILASP, which is also used in this paper, was employed to learn the ASP programs generated by LLMs. While LLMs are valuable tools for processing text and converting it into corresponding logical encodings, their primary limitation lies in the granularity of the generated encodings. In particular, a

*Research partially supported by Interdepartment Project on AI (Strategic Plan of UniUD-22-25)

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 405–419, doi:10.4204/EPTCS.439.28

© A. Dovier, T. Dreossi, A. Formisano & B. Strizzolo
This work is licensed under the
Creative Commons Attribution License.

common challenge with legal rules is the occurrence of distinct terms that share the same meaning. To address this issue, it is necessary to either distinguish between or unify these terms based on their meaning, as well as establish possible co-references within the text.

At the beginning Prolog was the language employed in legal reasoning; however due to its limited capabilities on commonsense/non-monotonic reasoning and in particular in handling exceptions, belief revision, default negation, ASP offers a more appropriate alternative [7]. As an example, Athakravi et al. [6] proposed a logic-based approach to generate a declarative model of judgments from past legal cases, capturing legal reasoning through rules and exceptions. In contrast, s(CASP), used by Sartor et al. [34], follows a top-down goal-oriented strategy. It includes an explanation tool that traces the paths leading to the solution; however, it does not provide a comprehensive model of the ASP encoding. Top-down approaches, however, are generally less suitable for legal reasoning due to scalability issues and may yield to conclusions even when applied to inconsistent programs, i.e. those that do not admit any stable model. Similarly, the tool developed by Arias et al. [4] uses s(CASP), and hence it must operate under the assumption that the system is consistent; as a matter of fact, it may return an answer even when the program does not admit any stable model.

One key advantage of ASP is its explainability in contrast to black-box algorithms, which present four different issues: opacity, strangeness, unpredictability, and justification [9]. Notably, even individuals involved in the penal system, such as police officers, have expressed scepticism towards opaque tools [36]. An example of a widely discussed black-box system in the legal domain is COMPAS (Correctional Offender Management Profiling for Alternative Sanctions) [37], a proprietary risk assessment tool used in the United States to predict the likelihood of criminal reoffending. It has been heavily criticised for its lack of transparency and potential biases, particularly regarding race and socioeconomic status.

In this context, Inductive Learning of Answer Set Programs (ILASP), an *Inductive Logic Programming* (ILP) framework [26], offers a novel approach for learning legal reasoning, while ensuring interpretability. The reason why we chose this approach is that it employs ASP, allowing the development of a non-monotonic reasoning system. In this study, we exploit ASP, for encoding immutable legal rules, and a Conflict-Driven Clause Learning (CDCL) bottom-up ASP solver, to derive stable models. In previous work [16] we focused on the law encodings in particular on how dealing the notion of vagueness; in [11] the basis of the project later developed for this paper were presented. In this work, we finalized that studies by augmenting the set of encoded articles from the Italian penal code, applying ILASP (version 4) to derive rules explaining judges' reasoning patterns. Moreover, since ASP by itself produces stable models but does not show the rules that were triggered to get those results, as explained by [33], in this paper we propose the use of xASP [3]. XASP is a tool that allows us to interpret and understand the reasoning behind the stable models generated by the ASP solvers by constructing a *Directed Acyclic Graph* (DAG). The integration of ILP framework and xASP has already been proved to work in a similar study applied to a different field [18]. This approach enables the system to compute scenarios of plausible decisions, each accompanied by an explanation, which can then be reviewed by a judge to inform her/his final decision. Alternatively, it can be used to generate a logical explanation for decisions autonomously made by a judge. Although, in principle, the system could be used for autonomous decision-making, the authors would suggest its use as a decision support system.

The paper is organized as follows. In Section 2 we briefly recall the basic notions on ASP and ILP. Section 3 describes the main aspects of our ASP encoding of the Italian Criminal Code. Such encoding forms the basis for the tasks of prediction, explanation, and dynamic model refinement described in Sections 4 and 5. In Section 6 we briefly present the overall system and we draw some conclusions and lines for future work.

2 Preliminaries

We assume the reader is aware of the main logic programming concepts. We repeat here some of the definitions for the sake of readability.

2.1 ASP and Stable Model Semantics

Let us consider a denumerable set of variables $\mathscr{X}$, a set of constant symbols $\mathscr{C}$, and a set of predicate symbols $\mathscr{P}$. An atomic formula (atom) is a formula of the form $p(t_1,\ldots,t_n)$, where $p \in \mathscr{P}$, and $t_1,\ldots,t_n$ are constant symbols or variables. A literal is either an atomic formula or `not` A, where A is an atomic formula. A (general) *rule* is a formula of the form

$$H \leftarrow B_1,\ \ldots,\ B_m,\ \texttt{not}\ C_1,\ \ldots,\ \texttt{not}\ C_n \tag{1}$$

where H, B_i, C_j are atomic formulas. If $n = 0$, a rule is said *definite*; if $m = n = 0$, it is a *fact*.

Variables are implicitly universally quantified and, logically, rules are implications. A *program* P is a set of clauses. A literal, a rule, a program is *ground* if it contains no variables. Given a rule, its *grounding* is the set of all ground rules obtained replacing consistently all its variables with constant symbols. The grounding of a program P is the set of all ground rules obtained by grounding rules in P.

The semantics of a program is based on the notion of *answer set/stable model* [20]. Given a ground program P, a set of atoms M is an answer set of P if it is the unique minimum model of the so-called "reduct" P^M obtained from P by first removing all rules with `not`A in their body where $A \in M$ and then removing the negated literals from all remaining rules (let us remark that P^M is made by definite rules). A program may have no answer sets, one answer set, or multiple answer sets. If P admits a stable model we say that it is consistent. We define with $\models_{sm}$ the logical consequence under stable models, namely we denote with $P \models_{sm} A$ the fact that P is consistent and the atom A is true in all stable models of P, and with $P \models_{sm}$ `not`A the fact that P is consistent and the atom A is false in all stable models of P.

In addition to the rules of form (1), the ASP language has been extended over time with several constructs such as denials (rules without head H, that state that the body must be false in all stable models), cardinality constraints, and aggregates that will be used in the following section. Of particular interest are the choice rules ($\texttt{lb}\{A_1;\ldots;A_k\}\texttt{ub} \leftarrow B_1,\ \ldots,\ B_m,\ \texttt{not}\ C_1,\ \ldots,\ \texttt{not}\ C_n.$), which are rules that use disjunction in the head. The number of satisfied atoms in their head matches the specified cardinality bounds (`lb` and `ub`). For example, consider the following ASP program P constituted by just a single choice rule: `1{p;q}2`. The answer sets of P are: {`p`}, {`q`} and {`p`, `q`}. We refer the reader to the literature on ASP for a detailed treatment [13, 19, 27].

Although deciding whether a ground program P admits answer sets or not is NP-complete, efficient solvers for ASP are available. These tools search for answer sets of a given program P by proceeding in two phases. First, they compute the grounding of P, and then they look for answer sets of the ground version using several clever techniques such as conflict-driven clause (rule) learning [19].

2.2 Inductive Logic Programming

Inductive Logic Programming, briefly ILP [29], is a subfield of both machine learning and logic programming, that focuses on learning logical rules from examples and background knowledge. The goal is to find a hypothesis (i.e., a set of rules) that explains the given examples within the context of the background knowledge (again, a set of rules).

> *From inductive machine learning, ILP inherits its goal: to develop tools and techniques to induce hypotheses from observations (examples) and to synthesize new knowledge from experience. By using computational logic as the representation mechanism for hypotheses and observations, ILP can overcome the two main limitations of classical machine learning [...]*
>
> 1. *the use of a limited knowledge representation formalism [...], and*
> 2. *difficulties in using substantial background knowledge in the learning process.*
>
> *S.H. Muggleton and L. De Raedt [30]*

Logic Programming and in general Computational Logic focuses on deductive reasoning. From facts and rules, their logical consequences are computed. The activity of finding hypotheses, instead, has to do with induction rather than deduction:

> *Induction is, in fact, the inverse operation of deduction, and cannot be conceived to exist without the corresponding operation, so that the question of relative importance cannot arise. Who thinks of asking whether addition of subtraction is the more important process in arithmetic? But at the same time much difference in difficulty may exist between a direct and inverse operation.* *W.S. Jevons [21]*

2.3 ILP Learning tasks

A *learning problem (learning task)* is defined as a triple $T = \langle B, S, E \rangle$, where

1. B is an ASP program called *background knowledge*
2. S is a set of ASP rules called *hypothesis space*, and
3. E is a set of *examples*.

This general schema admits some variants, in particular in the form of rules admitted in B and in E and in the set of examples (that can be positive, negative, restricted only to facts or involving more general rules). The overall idea is that of finding a (minimal) subset $H \subseteq S$ such that

$$B \cup H \text{ “explains” the examples in } E \tag{2}$$

The informal notion *explains* should be made explicit, but it depends on the kind of rules admitted for B, S, on the kind of examples considered, and on our requirements for a *good* explanation.

ASP was formally defined in the late Nineties [28], in particular with the availability of the first ASP solver SMODELS [31]. Before that, ILP focused on definite clause programs (basically, the same admitted by the Prolog language [23]). Let us focus directly into the more complex case of (general) ASP programs where a *learning problem* $T = \langle B, S, E \rangle$ is composed by B and S (general) ASP programs, and E is made by two parts: E^+ which is a set of ground facts that should be entailed, and E^- which is a set of ground facts that should be not. The lack of monotonicity of the language of general ASP programs leaves (at least) two formal approaches for the informal notion of *explanation*.

Brave Reasoning We look for a subset $H \subseteq S$ such that there is *at least one* stable model M of $B \cup H$ such that $E^+ \subseteq M$ and $E^- \cap M = \emptyset$. In practice, we look for a program where at least one of its stable models is coherent with both the positive and the negative requirements coming from the known examples E, namely a stable model of $B \cup H \cup \hat{E}$ where $\hat{E} = \{:- \texttt{not}\, A \,|\, A \in E^+\} \cup \{:- A \,|\, A \in E^-\}$ is a set of denials.

Algorithm 1 Cautious computation of H, given B and $E = \langle E^+, E^- \rangle$

```
for all H ∈ ℘(S) do                                   ▷ Start from ∅ and add elements
    C ← True
    repeat
        M ← Solver(B ∪ H)
        if (M ≠ False) ∧ ((E⁺ ⊈ M) ∨ (E⁻ ∩ M ≠ ∅)) then
            C ← False
    until M = False ∨ C = False
    if C then
        return H
return False
```

Cautious Reasoning We look for a subset $H \subseteq S$ such that *for all stable models* M of $B \cup H$ it holds that $E^+ \subseteq M$ and $E^- \cap M = \emptyset$. Using the notion of logical consequence under stable models, this can be restated as $B \cup H \models_{sm} E$. This can be computed calling a complete ASP solver `Solver` that returns a (different) stable model per call, and if there is not a (another) stable model, as sketched in Algorithm 1.

2.4 ILASP

ILASP [26] is a learning framework capable of solving more complex learning tasks than those needed for the scope of this paper. In particular, examples may be context-dependent, and errors in the input data (noisy data) can be handled with a notion of penalty.

Each encoded example in ILASP consists of three sets of ground atoms: inclusions, exclusions, and context. Considering the context C, every example has a broader background knowledge. In particular, $B \cup H \cup C$ must *cover* all examples. Given any framework ILP_X, learning task T_X and example e, we say that e is covered by a hypothesis if the hypothesis meets all conditions that T_X imposes on e. Specifically, at least one answer set must exist for each example, containing all inclusions while omitting all exclusions.

To describe the hypothesis space, various approaches can be employed. The first one is *direct rule declaration*, which explicitly specifies rules to define the hypothesis space: each rule in the list must be preceded by its *length*, representing the number of literals or components that make up the rule, followed by the symbol $\sim$, indicating the belonging of this rule to the hypothesis space. The second approach is *mode declarations*: ground atoms that contain placeholders such as `var(t)` or `const(t)`, that can be substituted with any variable or constant of type `t`, respectively. To declare head predicates, the notation `#modeh(pred(.))` is used, or alternatively, `#modeha(pred(.))` if aggregates are permitted. For body predicates, `#modeb(n, pred(.))` is employed, where `n` defines the maximum number of times `pred(.)` can appear in the body. Additionally, conditions can be specified using `#modec(c)`, where `c` represents a logical condition, such as `var(t1)>var(t2)`. To reduce the hypothesis space size and the time spent to compute it, specialized constants can be introduced within the different kinds of mode declarations. The constant `positive` restricts the search by excluding negation as failure of the predicate from the mode declaration. Additional constraints, such as `symmetric` and `anti_reflexive`, can be applied to binary predicates. In particular, when a predicate is marked with `symmetric`, the program recognizes `p(A,B)` and `p(B,A)` as equivalent, thus considering only one rule with this predicate. The constant `anti_reflexive` prevents reflexivity, that is, predicates `p(A,A)` are not considered. Finally, to further refine the hypothesis space, predicates such as `#maxv` can be used to specify the maximum number

of variables occurrences in any rule generated. To clarify the differences between the two approaches, consider this example:

```
% 1st approach: direct rule declarations
5 ~ damage(R, V) :- slap(R, V), agent(V), agent(R), R!=V.
% 2nd approach: mode declarations
#modeh(damage(var(agent), var(agent)), (anti_reflexive, positive)).
#modeb(1, slap(var(agent), var(agent)), (anti_reflexive, positive)).
```

In the first approach, the rule based on a prior judgment is directly encoded into the program (the number 5 is its length). Conversely, in the last approach, the predicates that may appear in the head and body of the rule to be learned are specified without encoding the entire rule. In the example above, we do not explicitly add the constant `symmetric`, as the two variables representing agents are distinct (i.e., they refer to the roles of "reo", that is the offender, and "victim"). Consequently, the hypothesis space accounts for both cases, whether the variables are symmetric or not, but the system ultimately infers from the examples provided in the program that they are distinct.

3 The Italian Criminal Code and its encoding in ASP

3.1 ASP encoding

To create a background knowledge for ILASP, some articles of the ICC[1] have been manually encoded in ASP. The articles were chosen according to our first work [17], which was focused on encoding vague concepts. Vagueness is present in several articles of the ICC; for our case study, we chose to focus on the vagueness involved in the definitions of theft and theft by snatching. Another instance of vagueness, that is the level of *bruise*, emerged during the encoding of articles related to assault and personal injury. ILP was not used to infer these rules, as they are immutable and must be precisely defined. For example, the rule "*if you subtract an object from another person by snatch, then you are guilty of snatch theft*" was translated as follows:

```
theft_snatch(R, V) :- theft(R, V, C), res_violence(R, C),
                      not person_violence(R, V).
```

The last literal represents the exceptions to this verdict; indeed, its meaning is that a case qualifies as snatch theft unless the snatching has caused violence toward the person, in which case the judgment changes to robbery.

The encoding in ASP could have been made using LLMs; however, these models still exhibit weaknesses and low sensitivity to the nuances of legal language. For instance, when prompted to encode the concept of an object's physical adherence to a victim's body (see the following paragraph), ChatGPT produces an output that clearly fails to capture the precise legal meaning we intend to express:

```
% Tight adherence:   theft is complete as soon as
%     the thief establishes effective control over the object.
tight_adherence(P, X) :- establishes_control(P, X).

% Tight adherence implies physical adherence but
%     focuses on the immediate establishment of control.
establishes_control(P, X) :- physical_adherence(P, X).
```

[1] https://www.gazzettaufficiale.it/sommario/codici/codicePenale

Even if the encoding was performed manually, we tried to keep homogeneity in the predicates. This means that if the same concept, such as "violence", appears in more than one article, then it must be encoded with the same predicate. Since legal terms have many shades, this work required also discussing with experts of the juridical field and this part was presented in [16]. Furthermore, also to reduce the vocabulary size, either predicates and variables (for ILASP) are inflected in the singular, as for nouns. Instead, verbs encoding criterion is to conjugate them in singular and present tense.

This part of encoding, at the beginning, comprehended only the encoding of articles, without considering any previous judgment. By the time we learned new rules from ILASP, those rules were added to the set after being "paired" with a marker predicate to distinguish them from the ones deriving directly from articles (see Section 5.1). The reason behind this is that, in Italy, old judgments can be used just as suggestions, but they do not become part of the legal code. Adding the marker predicate allowed us to inform users whether the outcome was based on previous judgments or just on articles.

3.2 Vagueness handling

Vague concepts are very common in legal texts and therefore must be taken into account when encoding laws. This study addresses vague concepts by exploiting non-deterministic choices. In other words, we assign to a vague concept a range of possible meanings to choose from. For example, as presented in [17], the level of adherence of an object to the victim's body is decisive to distinguish between robbery and snatch theft. Unfortunately, this level of adhesion is not objectively describable in any way. In such cases of non-determinacy, choice rules can be applied:

```
1{adherence(V,C,L):level(L)}1 :- unknown_adherence(V,C),
                                 subtracted_obj(C), victim(V).
```

Those kind of rules lead to multiple answer sets, specifically one for each level of adherence. During the procedural trial, the tool can also be used to explain the reasoning behind a judge's decision. New evidence or information can then be integrated to refine these justifications by introducing denials. For instance, if a photograph demonstrates that there was sufficient space between the victim's handbag and their waist, the ASP constraint that follows can be added to reflect this fact:

```
:- adherence(V,C,L), level(L), L < 2, subtracted_obj(C), victim(V).
```

3.3 Model verification and static refining

As result of the previous stages, we obtain an ASP program, P_{law}, which encodes juridical principles. To enhance the quality of this model, a refinement step can be performed.

A set of known criminal cases is retrieved from the databases Foroplus and DeJure.[2] Each case C is composed by a set (conjunction) of atoms. Let us focus on a case $C = \{A_1, \ldots, A_k\}$, then,

- For $i = 1, \ldots, k$, run the ASP solver on the program $P_{law} \cup \{A_i\}$. If no stable model is found (inconsistency), it means that the rules in P_{law} are too restrictive. In such cases, the issue is examined and the program is refined accordingly by adding rules that introduce possible exceptions. This is typically achieved by relaxing strict rules through the addition of literals such as `not exception(.)`, where `exception` is a newly introduced predicate that accounts for exceptional scenarios.
- Otherwise (a stable model exists), a further verification step is performed. It consists in adding a denial $\leftarrow$ `not` A_j for each atom A_j of C_i. The new program is input to the ASP solver to check

[2] `https://www.foroplus.it/home`, `https://dejure.it`

whether the expected model is obtained. If it is not, this suggests that some rules in P_{law} are too weak and again it needs refinement.

Once the model behaves as expected, a final verification can be conducted to determine whether different models arise when only a subset of C_i facts is assumed to hold. This process can reveal potential exceptions that extend P_{law}, refining the legal knowledge encoded in the system.

3.4 Contradictions' handling

During the manual encoding process, contradictory rulings were encountered. To address these discrepancies, an initial approach was to prioritize them through the application of the so-called "*Massime di Interpretazione*", principles designed to interpret and resolve conflicting arguments [5]. These maxims are divided into three categories:

- "*Lex specialis*": specific rulings take precedence over more general ones.
- "*Lex superior*": norms issued by higher authorities are given priority.
- "*Lex posterior*": recent verdicts have priority over earlier ones.

In some cases, it was not possible to establish a priority based on these principles, as they were in conflict with one another. To highlight these inconsistencies, an additional predicate, `contradiction`, was introduced. Thus, when two conflicting verdicts arise - with the facts supporting one offense while an alternative judgment suggests another - the predicates within the answer set will denote the existence of a contradiction. For instance, a ruling (Tribunale Bari sez. I, 26/08/2022, n. 3684) asserts that "cervicalgia"[3] cannot be considered a medical condition, but merely "pain". This contrasts with an earlier judgment by a higher authority (Cassazione penale sez. II, 12/03/2008, n. 15420), which classifies "cervicalgia" to be an illness. In this case, the two rulings are in contrast because the first one should have priority due to "Lex posterior", while the second one should take precedence for "Lex Superior". In the first case, if the harm caused by the offender is identified as "cervicalgia", the crime cannot be categorized as "personal injuries". However, in the second case, it does, since "cervicalgia" is considered an illness. This contradiction can be addressed as follows:

```
contradiction("not illness", "illness", I) :- only_pain(I), illness(I).
```

Another case of contradiction emerged when sentences were inconsistent with the legal articles themselves. The following example concerns the crime of personal injuries caused in conjunction with snatch theft. According to Articles 624 bis and 628, the distinction between this crime and robbery lies in the fact that, in the latter, violence or threats are directed against the victim. This violence also corresponds to the one involved in beatings and personal injuries (Tribunale Nocera Inferiore, 23/06/2020, n. 551). Therefore, personal injuries should be related to robbery rather than snatch theft. According to the encodings of the articles, the facts activate the rule whose head is `robbery`, but the actual judgment classifies it as snatch theft. When both predicates are present, a contradiction arises, triggering the `contradiction` predicate.

4 Predictions with explanations

Once the ASP solver is run on a program P_{law} and a sentence C added as denial (see Section 3.3), the resulting stable model can be analyzed to extract the reasoning behind the outcome. The tool can be

[3] *neck pain or cervical pain*

utilized in two modes. Firstly, it can explain the reasoning behind independent judicial decisions, in order to provide a clearer understanding of how the verdicts are reached. Secondly, its capability of generating different possible consistent scenarios, in the case of vague legal concepts, can be exploited.

Two approaches for ASP explainability have been investigated. The first one makes use of `xclingo2`, which allows users to annotate rules and facts in order to get justifications. Indeed, its key strength lies in its ability to generate explanations for why certain atoms appear in an answer set, allowing rules and atoms to be described in natural language. We report an example: consider the following program

```
cause("Carlo", "Beatrice", "skin lesion").
physical_illness("skin lesion").
intent_to_harm("Carlo", "Beatrice").
```

Examples of annotations (the ones starting with `%!`) we made in `xclingo2` are the following:

```
%!trace {"% is an illness", Z} illness(Z).
%!trace {"% is a physical illness", Z} physical_illness(Z).
illness(Z) :- physical_illness(Z).
%!trace {"It is evident that % (perpetrator) caused injuries to % (victim)", X,
↪  Y} injuries(X, Y).
injuries(X, Y) :- cause_illness(X, Y), intent_to_harm(X, Y), agent(X), agent(Y).
%!trace_rule {"% caused % to suffer %", X, Y, Z}
cause_illness(X, Y) :- cause(X, Y, Z), illness(Z), agent(X), agent(Y).
```

Those annotations produced the following explanation:

```
|__It is evident that Carlo (perpetrator) caused injuries to Beatrice (victim)
|  |__Carlo caused Beatrice to suffer skin lesion
|  |  |__skin lesion is an illness
|  |  |  |__skin lesion is a physical illness
|  |  |__Carlo caused skin lesion to Beatrice
|  |__Carlo had general intent to harm Beatrice
```

This output represents a structured justification tree explaining why Carlo has been found guilty of causing injury to Beatrice. In fact, the tree structure represents a chain of reasoning, where each indented line explains or justifies the statement above it. The average time required to get this kind of explanations is 0.078 s (30 test cases). In particular, it emerged that explanations for a negative verdict, that is when there is no evidence of the crime, required around 0.01 s more than the positive ones.

The other approach utilizes xASP, which provides a visual explanation of the outcome, representing the answer set by a DAG. For example, in the judgment (Cassazione penale sez. II, 21/02/2019, n. 16899), the court classified as robbery the defendant's actions of holding the victim still by pressing her head while taking her earrings. This scenario can be encoded as follows, assuming that the adherence of the object was tight since earrings are locked to the earlobes:

```
own("Veronica", "earrings").
subtract("Giulio", "earrings").
snatch("Giulio", "earrings").
take_possession("Giulio", "earrings").
adherence("Veronica", "earrings", 4).
```

In the graph shown in Fig. 1, nodes represent (inferred) facts, while arrows indicate the rules applied to derive conclusions. Specifically, each arrow points to the nodes that support its inference. The average time required to get the explanation graph was of 1.65 s (30 test cases).

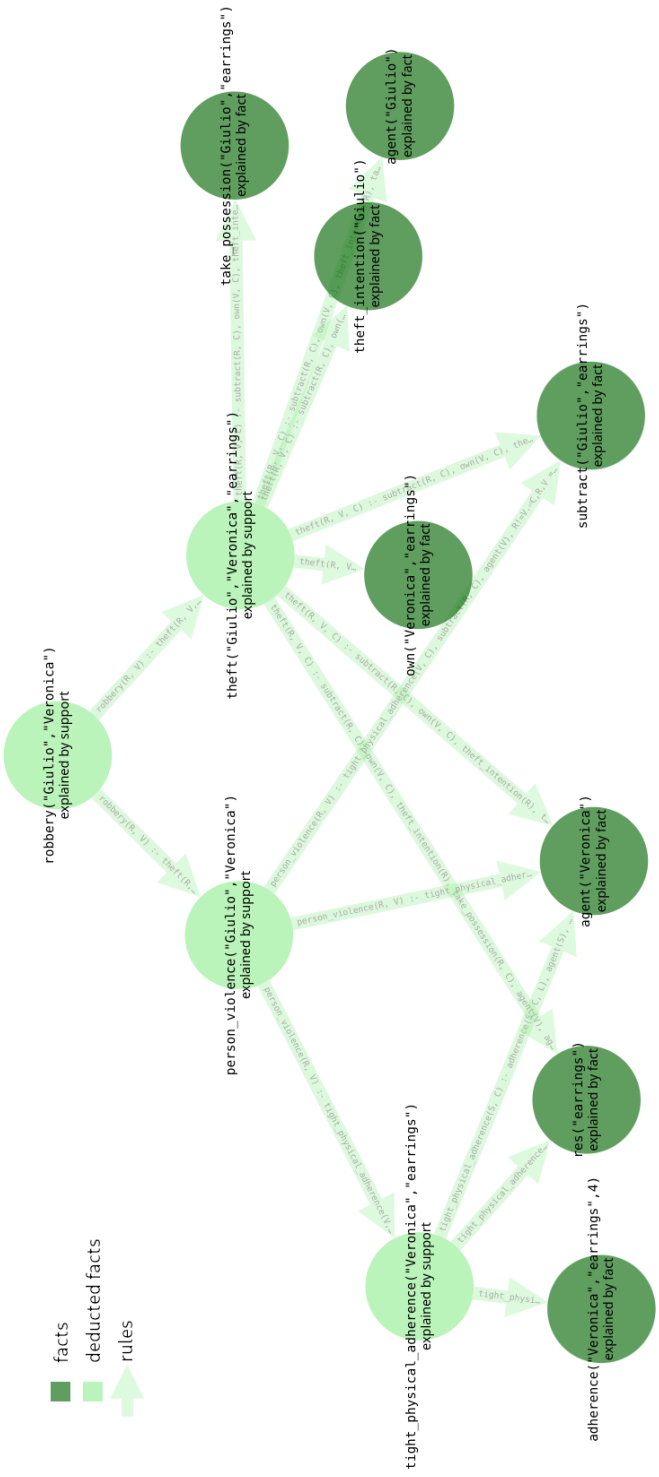


Figure 1: An example of a DAG produced by xASP to explain inferences (cf., Section 4). The colors are those used by xASP: dark green represents the facts, while light green indicates inferred facts. The edges show the rule used in the inferences and indicate the facts on which they depend.

5 Dynamic Model refining

Since new sentences on crimes are made anytime, it is useful to implement a module, preferably automated, to continuously refining the tool when adding this new knowledge to P_{law}. We employed ILASP (Section 2.4) that learns new rules from the new sentences and extends P_{law} accordingly. The manual intervention may be necessary in cases where contradictions arise.

5.1 ILASP encoding

In order to learn legal rules, there were manually encoded examples of previous judgments. The variables were instantiated according to the verdicts, taken from two legal databases Foroplus and DeJure used (see Section 3). The terms belong to the specific domain of the problem/crime; for example, personal injuries involve medical conditions such as "ecchimosi",[4] "ematoma",[5] etc.

In each example, the inclusions specify the crimes of which the offender has been accused, and the fundamental predicates related to the verdict. The exclusions define the predicates that cannot appear in the corresponding answer set. Finally, the context encodes the facts regarding the specific judgment. An example of a judgment encoded in the program is reported below:

```
#pos({beatings("R", "V"),       % inclusions
      damage("R", "V")},
     {derive_illness("V")},     % exclusions
     {harmful_intention("R").   % context
      slap("R", "V").}).
```

There were used both brave and cautious reasoning, respectively in positive (`#pos`) and negative (`#neg`) examples. The first ones must be included in at least one answer set, while the negative ones cannot be true in any answer set.
The hypothesis space was explicitly defined encoding rules, each of them preceded by a number specifying the number of predicates involved, and the symbol $\sim$ representing the inclusion of the rule in the hypothesis space. Alternatively, there were used mode declarations, in which only the predicates that can appear in the head and in the body of the rule are specified. The rule learned from the example mentioned above is the following:

```
damage(R, V) :- R != V; agent(R); agent(V); slap(R, V).
```

Then, the learned rules have been added to the background knowledge. To distinguish the articles of the ICC from the rules learned from previous judgments, a cardinality constraint was used. Specifically, the learned rules are encoded in such a manner that, when the body of the rule is activated, the answer set includes both the original predicate learned and a predicate `using_judgment` which denotes that this verdict is referred to previous judgments.

5.2 Experimental Results

We ran ILASP (version 4) on different sets of encoded legal articles listed below:

- **Set**$_{AC}$: 583
- **Set**$_{CP}$: 575, 579, 584, 588, 589, 589 bis, 59, 595, 609 bis, 610, 614

[4] *ecchymosis*
[5] *hematoma*

- **Set**$_{B,PI}$: 581, 582
- **Set**$_{T,R}$: 624, 624 bis, 628

All tests were conducted on a system with an Intel 12th Gen Core i7-12700 processor, RAM of 16 GB and a maximum clock speed of 4.9 GHz. Its operating system is Ubuntu (24.04.2 LTS). The version of clingo used is 5.6.2.

For each set we encoded around 22 judgments as examples (except for the biggest set Set$_{CP}$, which involved 70 examples). The dimension of hypothesis space, denoted by $|S|$, corresponds to the number of rules and is reported in Table 1. Some of these sets, marked with *, employ mode declarations to generate the hypothesis space, while others rely on a predefined space. The use of mode declarations allows for a wider range of possible rules but comes at a higher computational cost. Indeed, for the sets that do not utilize mode declarations, incorporating them would render the learning process infeasible due to exceeding memory size. This challenge arises because, in the legal domain, laws are highly specific and introduce a large number of predicates, making the automatic generation of a suitable hypothesis space particularly demanding. In some cases, we adopted a hybrid approach, leveraging both techniques in a compensatory way. While some rules were manually encoded, others, typically the more general ones, were generated through mode declarations. This approach allows to balance the expressiveness of the hypothesis space with computational feasibility.

Table 1: Timing breakdown (in seconds) for different ILASP stages across datasets

	Set$^*_{AC}$	**Set**$_{CP}$	**Set**$_{B,PI}$	**Set**$^*_{B,PI}$	**Set**$_{T,R}$
$\|S\|$	5418	531	21	93996	11
Pre-processing	0.016	0.028	0.023	0.046	0.014
Hypothesis space generation	0.294	0.102	0.009	204.179	0.005
Conflict analysis	4.289	1.876	0.087	291.992	0.076
Counterexample search	0.048	0.676	0.099	0.179	0.088
Hypothesis search	1.372	0.311	0.022	52.092	0.010
Total	**6.075**	**3.005**	**0.240**	**549.576**	**0.196**

The results shown in Table 1 highlight the significant computational cost of mode declarations. For instance, in Set$^*_{B,PI}$, hypothesis space generation and conflict analysis require substantially more time (204.18s and 291.99s, resp.) compared to predefined rule-based configurations. In contrast, predefined rule-based sets, such as Set$_{B,PI}$ and Set$_{T,R}$, exhibit much lower execution times across all stages, demonstrating better efficiency but at the cost of reduced rule flexibility.

6 Conclusions

In this work, we have analyzed how to encode a legal knowledge base in ASP, how to use it for legal reasoning and for explaining the decisions made, and how to refine it almost automatically. The developed system is free and available from `http://clp.dimi.uniud.it/sw/`. Two screenshots are reported in Figure 2. The main page provides links to articles and judgments (in Italian), with each article's ASP encoding presented alongside its textual counterpart. Examples of previous verdicts analyzed by ILASP and then incorporated in the knowledge base are organized according to the articles they pertain to.

Figure 2: Homepage of the website (left), judgments of the encoded articles (right)

The decisions made by courts of the Italian criminal system are made at three distinct levels: 1. *primo grado*, 2. *appello*, 3. *cassazione*. At each level, the ruling can be modified depending on new proofs or by checking procedural errors of the previous level. We have developed a simple tool that rewrites the ASP encoding from the first level when new evidence is presented. This tool simulates the second level by updating facts. As future work we will deal with the third level.

The main advantage in adopting logic programming, compared to, for example, using ML-based techniques, is the immediate explainability properties of the system. Moreover, in the legal domain the size of the data can still be handled by an ASP program (this is different from other domain families such as, for example, the weather forecast domain, where data is computed, stored, and analyzed for each second and for all cubic meters of the atmosphere of the planet earth).

A judge might be interested in analyzing all stable models obtained. If a property holds in all stable models, it is reasonable to believe in it, thus with a sort of cautious reasoning, it can be proposed and accepted by humans. If it holds in one/few stable models, it might be temporarily accepted (a sort of brave reasoning), but maybe it requires further analysis.

The choice of ILASP as inference engine for automatizing this part opens further lines of research. For instance, we intend to apply to the ILASP solver the ASP-technology developed in parallelizing ASP-solvers [12, 13, 14, 15] in order to improve efficiency and scalability. Concerning the explainability issues we will relate our results to those produced by argumentation-based approaches [32].

On the other hand, a key limitation is that, although ILASP is capable of handling dynamic updates, the initial knowledge representation still relies on human expertise to translate legal texts into ASP rules. We acknowledge that this manual encoding process can become a bottleneck, and we therefore plan to address this in future work by exploring methods for its automation.

References

[1] L.E. Allen (1957): *Symbolic logic: a razor-edged tool for drafting and interpreting legal documents*. *The Yale Law Journal* 66(6), pp. 933–879, doi:10.2307/794073.

[2] L.E. Allen & C.S. Saxon (1987): *Some Problems in Designing Expert Systems to Aid Legal Reasoning*. In: *Proc. of ICAIL'87*, ACM, pp. 94–103, doi:10.1145/41735.41747.

[3] M. Alviano, L.L.T. Trieu, T.C. Son & M. Balduccini (2024): *The XAI system for answer set programming xASP2*. *J. Log. Comput.* 34(8), pp. 1500–1525, doi:10.1093/LOGCOM/EXAE036.

[4] J. Arias, M. Moreno-Rebato, J.A. Rodríguez-García & S. Ossowski (2024): *Automated legal reasoning with discretion to act using s(LAW)*. *Artif. Intell. Law* 32(4), pp. 1141–1164, doi:10.1007/S10506-023-09376-5.

[5] K.D. Ashley (2017): *Artificial Intelligence and Legal Analytics: New Tools for Law Practice in the Digital Age*. Cambridge University Press, doi:10.1017/9781316761380.

[6] D. Athakravi, K. Satoh, M. Law, K. Broda & A. Russo (2015): *Automated Inference of Rules with Exception from Past Legal Cases Using ASP*, p. 83–96. Springer Int. Publishing, doi:10.1007/978-3-319-23264-5_8.

[7] C. Baral & M. Gelfond (1994): *Logic programming and knowledge representation*. *The Journal of Logic Programming* 19, pp. 73–148, doi:10.1016/0743-1066(94)90025-6.

[8] M.A. Borroto Santana, I. Kareem & F. Ricca (2024): *Towards Automatic Composition of ASP Programs from Natural Language Specifications*. In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, ijcai.org, pp. 6198–6206, doi:10.24963/ijcai.2024/685.

[9] B. Brożek, M. Furman, M. Jakubiec & B. Kucharzyk (2024): *The black box problem revisited. Real and imaginary challenges for automated legal decision making*. *Artificial Intelligence and Law* 32(2), pp. 427–440, doi:10.1007/s10506-023-09356-9.

[10] E. Coppolillo, F. Calimeri, G. Manco, S. Perri & F. Ricca (2024): *LLASP: Fine-tuning Large Language Models for Answer Set Programming*. In P. Marquis, M. Ortiz & M. Pagnucco, editors: *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning, KR 2024, Hanoi, Vietnam. November 2-8, 2024*, pp. 834–844, doi:10.24963/KR.2024/78.

[11] A. Dovier, T. Dreossi & A. Formisano (2024): *XAI-LAW Towards a Logic Programming Tool for Taking and Explaining Legal Decisions*. In: *Proceedings of the 39th Italian Conference on Computational Logic, Rome, Italy, June 26-28, 2024*, *CEUR Workshop Proceedings* 3733, CEUR-WS.org. Available at `https://ceur-ws.org/Vol-3733/short3.pdf`.

[12] A. Dovier, A. Formisano, G. Gupta, M.V. Hermenegildo, E. Pontelli & R. Rocha (2022): *Parallel Logic Programming: A Sequel*. *Theory Pract. Log. Program.* 22(6), pp. 905–973, doi:10.1017/S1471068422000059.

[13] A. Dovier, A. Formisano & E. Pontelli (2018): *Parallel Answer Set Programming*. In Y. Hamadi & L. Sais, editors: *Handbook of Parallel Constraint Reasoning*, Springer, pp. 237–282, doi:10.1007/978-3-319-63516-3_7.

[14] A. Dovier, A. Formisano, E. Pontelli & F. Vella (2016): *A GPU Implementation of the ASP Computation*. In M. Gavanelli & J.H. Reppy, editors: *Proc. of PADL 2016*, *LNCS* 9585, Springer, pp. 30–47, doi:10.1007/978-3-319-28228-2_3.

[15] A. Dovier, A. Formisano & F. Vella (2019): *GPU-Based Parallelism for ASP-Solving*. In P. Hofstedt, S. Abreu, U. John, H. Kuchen & D. Seipel, editors: *Declarative Programming and Knowledge Management*, *LNCS* 12057, Springer, pp. 3–23, doi:10.1007/978-3-030-46714-2_1.

[16] M. Dozzi, T. Dreossi, L. Baron, F. Costantini, A. Dovier & A. Formisano (2024): *Semi-automatic knowledge representation and reasoning on vague crime concepts*. Workshop DigForASP, colocated with 15th European Symposium on Computational Intelligence and Mathematics. Krakow (Poland). May 12th-15th, 2024.

[17] M. Dozzi, T. Dreossi, F. Costantini, A. Dovier & A. Formisano (2023): *Semi-automatic knowledge representation and reasoning on vagueness crime concepts*. ALP2023, JURIX 2023 workshop on AI, law and philosophy, Maastricht, Netherlands.

[18] T. Dreossi, A. Dovier, A. Formisano, M. Law, A. Manzato, A. Russo & M. Tait (2025): *Towards Explainable Weather Forecasting Through FastLAS*. In C. Dodaro, G. Gupta & M.V. Martinez, editors: *Logic Programming and Nonmonotonic Reasoning*, Springer Nature Switzerland, Cham, pp. 262–275, doi:10.1007/978-3-031-74209-5_20.

[19] M. Gebser, R. Kaminski, B. Kaufmann & T. Schaub (2012): *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, doi:10.2200/S00457ED1V01Y201211AIM019.

[20] M. Gelfond & V. Lifschitz (1988): *The stable model semantics for logic programming*. In: *ICLP/SLP*, 88, Cambridge, MA, pp. 1070–1080, doi:10.2307/2275201.

[21] W. Jevons (1874): *The Principles of Science: A Treatise on Logic and Scientific Method*. Macmillan, London, doi:10.5962/bhl.title.31805.

[22] I. Kareem, K. Gallagher, M. Borroto, F. Ricca & A. Russo (2024): *Using Learning from Answer Sets for Robust Question Answering with LLM*. In C. Dodaro, G. Gupta & M.V. Martinez, editors: *Logic Programming and Nonmonotonic Reasoning*, Springer, Cham, pp. 112–125, doi:10.1007/978-3-031-74209-5_9.

[23] R.A. Kowalski (1974): *Predicate Logic as Programming Language*. In J.L. Rosenfeld, editor: *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974*, North-Holland, pp. 569–574. Available at `https://www.doc.ic.ac.uk/~rak/papers/IFIP%2074.pdf`.

[24] R.A. Kowalski (1992): *Legislation as logic programs*. In: *Proc. of LPSS'92*, Springer, pp. 203–230, doi:10.1007/3-540-55930-2_15.

[25] R.A. Kowalski, J.A. Dávila, G. Sartor & M. Calejo (2023): *Logical English for law and education*. In D.S. Warren, V. Dahl, T. Eiter, M.V. Hermenegildo, R.A. Kowalski & F. Rossi, editors: *Prolog: The Next 50 Years*, *LNCS* 13900, Springer, pp. 287–299, doi:10.1007/978-3-031-35254-6_24.

[26] M. Law, A. Russo & K. Broda (2020): *The ILASP system for Inductive Learning of Answer Set Programs*. *CoRR* abs/2005.00904, doi:10.48550/arXiv.2005.00904.

[27] V. Lifschitz (2019): *Answer Set Programming*. Springer, doi:10.1007/978-3-030-24658-7. ISBN:978-3-030-24657-0.

[28] V.W. Marek & M. Truszczynski (1999): *Stable Models and an Alternative Logic Programming Paradigm*. In K.R. Apt, V.W. Marek, M. Truszczynski & D.S. Warren, editors: *The Logic Programming Paradigm - A 25-Year Perspective*, Artificial Intelligence, Springer, pp. 375–398, doi:10.1007/978-3-642-60085-2_17.

[29] S.H. Muggleton (1991): *Inductive Logic Programming*. *New Gener. Comput.* 8(4), pp. 295–318, doi:10.1007/BF03037089.

[30] S.H. Muggleton & L. De Raedt (1994): *Inductive Logic Programming: Theory and Methods*. *J. Log. Program.* 19/20, pp. 629–679, doi:10.1016/0743-1066(94)90035-3.

[31] I. Niemelä & P. Simons (1997): *Smodels - An Implementation of the Stable Model and Well-Founded Semantics for Normal LP*. In J. Dix, U. Furbach & A. Nerode, editors: *Proc. of LPNMR'97*, *LNCS* 1265, Springer, pp. 421–430, doi:10.1007/3-540-63255-7_32.

[32] H. Prakken & G. Sartor (2015): *Law and logic: A review from an argumentation perspective*. *Artif. Intell.* 227, pp. 214–245, doi:10.1016/J.ARTINT.2015.06.005.

[33] L. Robaldo, S. Batsakis, R. Calegari, F. Calimeri, M. Fujita, G. Governatori, M.C. Morelli, F. Pacenza, G. Pisano, K. Satoh, I. Tachmazidis & J. Zangari (2023): *Compliance checking on first-order knowledge with conflicting and compensatory norms: a comparison among currently available technologies*. *Artif. Intell. Law* 32(2), p. 505–555, doi:10.1007/s10506-023-09360-z.

[34] G. Sartor, J.A. Dávila, M. Billi, G. Contissa, G. Pisano & R.A. Kowalski (2022): *Integration of Logical English and s(CASP)*. In J. Arias, R. Calegari, L. Dickens, W. Faber, J. Fandinno, G. Gupta, M. Hecher, D. Inclezan, E. LeBlanc, M. Morak, E. Salazar & J. Zangari, editors: *Proceedings of the International Conference on Logic Programming 2022 Workshops co-located with the 38th International Conference on Logic Programming (ICLP) 2022, Haifa, Israel, July 31st - August 1st, 2022*, *CEUR Workshop Proceedings* 3193, CEUR-WS.org. Available at `https://ceur-ws.org/Vol-3193/paper5GDE.pdf`.

[35] M.J. Sergot, F. Sadri, R.A. Kowalski, F. Kriwaczek, P. Hammond & H.T. Cory (1986): *The British Nationality Act as a logic program*. *Commun. ACM* 29(5), p. 370–386, doi:10.1145/5689.5920.

[36] M. Simmler, S. Brunner, G. Canova & K. Schedler (2023): *Smart criminal justice: exploring the use of algorithms in the Swiss criminal justice system*. *Artificial Intelligence and Law* 31(2), pp. 213–237, doi:10.1007/s10506-022-09310-1.

[37] E. Yong (2018): *A Popular Algorithm Is No Better at Predicting Crimes Than Random People*. *The Atlantic*. Available at `https://www.theatlantic.com/technology/archive/2018/01/equivant-compas-algorithm/550646/`.

Formally Explaining Decision Tree Models with Answer Set Programming

Akihiro Takemura
National Institute of Informatics, Tokyo, Japan
atakemura@nii.ac.jp

Masayuki Otani
Tokyo Institute of Technology, Tokyo, Japan
otani.m.af@m.titech.ac.jp

Katsumi Inoue
National Institute of Informatics, Tokyo, Japan
inoue@nii.ac.jp

Decision tree models, including random forests and gradient-boosted decision trees, are widely used in machine learning due to their high predictive performance. However, their complex structures often make them difficult to interpret, especially in safety-critical applications where model decisions require formal justification. Recent work has demonstrated that logical and abductive explanations can be derived through automated reasoning techniques. In this paper, we propose a method for generating various types of explanations, namely, sufficient, contrastive, majority, and tree-specific explanations, using Answer Set Programming (ASP). Compared to SAT-based approaches, our ASP-based method offers greater flexibility in encoding user preferences and supports enumeration of all possible explanations. We empirically evaluate the approach on a diverse set of datasets and demonstrate its effectiveness and limitations compared to existing methods.

1 Introduction

The widespread adoption of decision tree models in machine learning can be attributed to their high accuracy and computational efficiency. However, more complex ensemble variants, such as random forests and gradient-boosted trees, tend to operate as black-box models, rendering their internal decision-making processes opaque to users [12]. This lack of transparency is particularly problematic in high-stakes applications, such as healthcare, finance, and criminal justice, where trust in model decisions hinges on a clear understanding of how predictions are made.

Explainability, often referred to as eXplainable AI (XAI), seeks to bridge this gap by making model behavior understandable [12]. In practice, however, producing faithful explanations is challenging due to the complexity of modern machine learning models. Naive or heuristic-based techniques often fall short because they fail to capture the full logical structure underlying the model's decisions [23, 19].

Well-known post-hoc explanation methods such as LIME [21], SHAP [18], and Anchors [22] rely on heuristic approximations, and can yield explanations that are not guaranteed to align with the model's true behavior [20]. In contrast, logic-based approaches offer rigorous guarantees and have been successfully applied to generate various types of explanations, including sufficient and contrastive ones [15, 16, 14, 4, 5, 3, 2]. These logic-based approaches have been largely based on techniques of Boolean satisfiability (SAT). However, SAT-based methods often lack flexibility, especially when it comes to incorporating user-defined preferences or supporting complex reasoning tasks like explanation enumeration.

In this work, we propose an alternative approach using Answer Set Programming (ASP), a declarative logic programming paradigm well-suited for combinatorial reasoning. We show how ASP can be

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 420–437, doi:10.4204/EPTCS.439.29

© A. Takemura, M. Otani & K. Inoue
This work is licensed under the
Creative Commons Attribution License.

leveraged to encode and compute different types of explanations, including sufficient, contrastive, majority, and tree-specific explanations, for decision tree models. The high expressiveness of ASP allows us to concisely capture various explanation types and flexibly incorporate constraints, making it a promising tool for formal explainability.

Our main contributions are as follows:

1. We develop ASP encodings for multiple explanation types, including sufficient, contrastive, majority, and tree-specific explanations.
2. We implement and evaluate our approach across 20 datasets, comparing its performance against existing SAT-based methods.
3. We analyze the ASP-based method's strengths and limitations, highlighting where it excels (majority explanation enumeration) and challenges (scalability to large forests).

This paper is organized as follows: Section 2 introduces the necessary background on decision tree models and ASP. Section 3 presents our ASP-based encodings for generating various types of explanations. Section 4 details the experimental setup and results comparing our method with SAT-based approaches. Section 5 discusses related work in the domain of explainable AI for decision tree models. Section 6 presents the conclusion. Additionally, the Appendix provides worked examples and visual illustrations of our method applied to decision tree, random forest, and boosted tree models.

2 Background

This section introduces the necessary concepts for the rest of the paper. We first review Boolean functions and their relationship to logical reasoning in the context of machine learning, and the structures of decision tree models. We then present a concise overview of ASP, the declarative framework used to encode and compute formal explanations in our approach.

2.1 Boolean Functions

Let $X_n = \{x_1, \ldots, x_n\}$ be a set of Boolean variables. A *Boolean function* is a mapping $f : \{0,1\}^n \to \{0,1\}$, which assigns each Boolean assignment $\boldsymbol{x} \in \{0,1\}^n$ a Boolean value. We say $\boldsymbol{x}$ is a *positive instance* if $f(\boldsymbol{x}) = 1$, and a *negative instance* if $f(\boldsymbol{x}) = 0$. A propositional formula over X_n is constructed using logical connectives: conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$). A *literal* L is either a variable x_i (positive literal) or its negation $\neg x_i$ (negative literal). A *term* is a conjunction of literals.

For an assignment $\boldsymbol{z} \in \{0,1\}^n$, $t_{\boldsymbol{z}} = \bigwedge_{i=1}^{n} x_i^{z_i}$ represents its corresponding term, where $x_i^0 = \neg x_i$ and $x_i^1 = x_i$. A term t_1 is a *subterm* of a term t_2 if the literals appearing in t_1 appear in t_2. For example, $x_1 \wedge x_3$ is a subterm of $x_1 \wedge \neg x_2 \wedge x_3$. A term t *covers* $\boldsymbol{z}$ if t is a subterm of $t_{\boldsymbol{z}}$, i.e., t is satisfied under $\boldsymbol{z}$ [2]. This captures the idea that t holds whenever the assignment $\boldsymbol{z}$ satisfies all literals in t. A term t is an *implicant* of a propositional formula f if $t \models f$, i.e., every assignment $\boldsymbol{z} \in \{0,1\}^n$ that satisfies t also satisfies f. A term t is a *prime implicant* of f if $t \models f$ and no proper subterm of t entails f.

2.2 Trees and Forests

We consider a finite set of attributes (or features) $\mathscr{A} = \{A_1, \ldots, A_n\}$, where each attribute A_i takes values from a domain D_i, which may be Boolean, categorical, or numerical. Decision tree learning algorithms create Boolean tests: threshold tests $(A_i \leq c)$ for numerical attributes and membership tests $(A_i \in S)$ for

categorical attributes. Let $L = \{L_1, \ldots, L_n\}$ denote the Boolean literals from tree node tests, where each L_i corresponds to a Boolean variable from X_n. An instance $\boldsymbol{z} = (v_1, \ldots, v_n)$ with $v_i \in D_i$ evaluates each Boolean variable x_i based on the tree node tests. The corresponding term is $t_{\boldsymbol{z}} = \bigwedge_{i=1}^{n} x_i^{z_i}$, where $z_i = 1$ if the Boolean test for x_i evaluates to true and $z_i = 0$ if it evaluates to false.

Decision Trees. A *decision tree* classifier over $\mathscr{A}$ is a binary tree T where internal nodes test Boolean conditions and leaves are labeled with classes in $\{0, 1\}$. Tree traversal starts at the root and follows left branches for true conditions and right branches for false conditions.

Random Forests. A *random forest* $RF = \{T_1, \ldots, T_m\}$ is a set of decision trees T_is, where its output is determined by majority vote [6]:

$$RF(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{m} T_i(\boldsymbol{x}) > \frac{m}{2}, \\ 0 & \text{otherwise.} \end{cases}$$

Regression Trees and Forests. A *regression tree* is similar to a decision tree, except leaves are labeled with real-valued outputs. A *forest* $F = \{T_1, \ldots, T_m\}$ consists of m regression trees with aggregated weight $w(F, \boldsymbol{x}) = \sum_{k=1}^{m} w(T_k, \boldsymbol{x})$.

Boosted Trees. A *boosted tree* $BT = \{T_1, \ldots, T_m\}$ for binary classification is a sequence of trees [9] (typically regression trees, as in gradient boosted trees such as XGBoost [7]) with prediction determined by the sign of cumulative weights.

$$BT(\boldsymbol{x}) = \begin{cases} 1 & \text{if } w(BT, \boldsymbol{x}) > 0, \\ 0 & \text{if } w(BT, \boldsymbol{x}) < 0, \end{cases} \quad \text{where } w(BT, \boldsymbol{x}) = \sum_{k=1}^{m} w(T_k, \boldsymbol{x}).$$

We define the size of a forest F, and a boosted tree BT, as $|F| = |BT| = \sum_{k=1}^{m} |T_k|$, where $|T_k|$ is the number of nodes in the tree T_k.

2.3 Answer Set Programming

Answer Set Programming (ASP) is a form of declarative programming oriented towards solving hard combinatorial problems. It allows one to encode a problem as a set of logical rules, and uses a solver to compute its solutions, called *answer sets*, based on the stable model semantics [11].

A typical ASP rule is written as:

$$\mathrm{a}_1 \texttt{ :- } \mathrm{a}_2, \ \ldots, \ \mathrm{a}_\mathrm{m}, \ \texttt{not } \mathrm{a}_{\mathrm{m}+1}, \ \ldots, \ \texttt{not } \mathrm{a}_\mathrm{n}.$$

Here, each $\mathrm{a_i}$ is a first-order atom, and `not` denotes *default negation*. A rule with only the head $\mathrm{a_1}$ is called a *fact*, while one without a head is an *integrity constraint*, used to eliminate undesired models.

Modern ASP systems, such as *clingo* [10], support expressive constructs that simplify problem modeling, for example:

- Conditional literals of the form $\{\mathrm{a(X)} : \mathrm{b(X)}\}$, which represent sets of atoms $\mathrm{a(X)}$ for all values of `X` satisfying $\mathrm{b(X)}$.
- Cardinality constraints of the form $s_1 \ \{\mathrm{a(X)} : \mathrm{b(X)}\} \ s_2$, which restrict the number of true literals in the set to be between s_1 and s_2.

- Aggregates such as `#count` and `#sum` allows one to count and sum the elements, respectively.

These constructs allow for compact and readable encodings of complex search spaces, making ASP particularly well-suited for formal explanations. Moreover, the `#heuristic` directive provides solver-level preferences for atom selection. For further details, see the *clingo* user guide.[1]

3 Explaining Decision Tree Models

This section presents ASP-based encodings for extracting different types of explanations from decision tree models. We consider four explanation types (sufficient, contrastive, majority, and tree-specific), across three model types (decision trees, random forests, and gradient-boosted trees). Each explanation type is formally defined, and its corresponding ASP encoding is provided.

Our ASP encodings preserve the logical semantics of decision tree models through faithful representation of structure and decision flow, with integrity constraints ensuring only valid explanations.

Table 1 summarizes the explanation types supported for each model type in this paper.

Model	Sufficient	Contrastive	Majority	Tree-Specific
Decision Tree	✓	✓	n/a	n/a
Random Forest	✓	✓	✓	n/a
XGBoost	-	-	n/a	✓

Table 1: Explanation types supported by our ASP encodings for each model type. "-" indicates not implemented; "n/a" indicates not applicable.

3.1 Decision Tree

We begin with decision trees, where each model is translated into a set of ASP facts that represent its structure and prediction logic. Each explanation type is defined in this subsection, along with its corresponding ASP encoding. The predicates introduced by our preprocessing script are listed in Table 2, where `node/3` and `pre_class/1` are instance-dependent (truth values depend on the specific instance), while structural predicates like `leaf_node/2` are instance-independent. Across our encodings, including Random Forest and Boosted Trees, we use the `next_node/2` predicate, which simulates path traversal from the root to a leaf. By conditionally reconstructing the decision flow based on selected literals, it enables the solver to check whether an explanation preserves or alters the prediction, mirroring how decision trees arrive at their output. See Appendix 2 and Program 8 for an example.

Predicate	Meaning
`node(X,L,B)`	Node `X` checks whether literal `L` has truth value `B` in the current instance.
`leaf_node(X,C)`	Node `X` is a leaf predicting class `C`.
`left_node(X,LX)`	Node `X` has left child `LX`.
`right_node(X,RX)`	Node `X` has right child `RX`.
`pre_class(P)`	The predicted class of the instance is `P`.

Table 2: Predicates used to encode decision tree models as ASP facts.

[1] https://github.com/potassco/guide/releases/

3.1.1 Sufficient Explanation from a Decision Tree

Definition 1 (Sufficient Explanation [2]) *Let T be a decision tree over Boolean variables $X_n = \{x_1,\ldots,x_n\}$, let f be the Boolean function represented by T, and let $\boldsymbol{x} \in \{0,1\}^n$ be an instance with term $t_{\boldsymbol{x}}$. A sufficient explanation for $\boldsymbol{x}$ to be classified 1 (resp. 0) given f is a subterm t of $t_{\boldsymbol{x}}$ such that t is a prime implicant of f (resp. $\neg f$).*

By definition, changing any literal in a prime implicant will change the output of the corresponding Boolean function. Based on this, the sufficient explanation can be obtained by finding the maximal set of literals that can be changed without affecting the output, then taking the complement. The rule `node(X,L,0..1)` (Line 2) allows conditional traversal in both directions, and the `invalid` predicate (Line 8) detects output changes.

Program 1: Sufficient Explanation from a Decision Tree

```
1  1 {selected_literal(L):node(X,L,B)}.
2  node(X,L,0..1) :- selected_literal(L),node(X,L,B).
3  next_node(LX) :- node(0,L,1),left_node(0,LX).
4  next_node(RX) :- node(0,L,0),right_node(0,RX).
5  next_node(LX) :- next_node(X),node(X,L,1),left_node(X,LX).
6  next_node(RX) :- next_node(X),node(X,L,0),right_node(X,RX).
7  class(C):-next_node(X),leaf_node(X,C).
8  invalid :- class(0),class(1).
9  :- invalid.   %% remove solutions that change the prediction
10 #heuristic selected_literal(L). [1,true]   %% prefer including more literals
11 #show selected_literal/1.
```

This encoding will find the maximal set of `selected_literal` by utilizing the `#heuristic` directive in *clingo* which will prioritize assigning `true` to `selected_literal`. See Appendix Example 1 for an example.

3.1.2 Contrastive Explanation from a Decision Tree

Unlike sufficient explanations, we are looking for a minimal set of literals that will change the output of the Boolean function.

Definition 2 (Contrastive Explanation [2]) *Let T be a decision tree over Boolean variables $X_n = \{x_1,\ldots,x_n\}$, let f be the Boolean function represented by T, and let $\boldsymbol{x} \in \{0,1\}^n$ be an instance with term $t_{\boldsymbol{x}}$. A contrastive explanation for $\boldsymbol{x}$ to be classified 1 (resp. 0) given f is a subterm t of $t_{\boldsymbol{x}}$ such that removing the literals in t from $t_{\boldsymbol{x}}$ yields a term that is not an implicant of f (resp. $\neg f$), and for every literal $l \in t$, removing all literals in t except l from $t_{\boldsymbol{x}}$ does not satisfy this condition.*

Thus, we allow for this change in the prediction by introducing the `valid` predicate (Line 8) which must not be false according to the integrity constraint on the next line.

Program 2: Contrastive Explanation from a Decision Tree

```
1  1 {selected_literal(L):node(X,L,B)}.
2  node(X,L,0..1) :- selected_literal(L),node(X,L,B).
3  next_node(LX) :- node(0,L,1),left_node(0,LX).
4  next_node(RX) :- node(0,L,0),right_node(0,RX).
5  next_node(LX) :- next_node(X),node(X,L,1),left_node(X,LX).
6  next_node(RX) :- next_node(X),node(X,L,0),right_node(X,RX).
7  class(C) :- next_node(X),leaf_node(X,C).
```

```
valid :- class(0),class(1).
:- not valid.   %% the output must change
#heuristic selected_literal(L). [1,false]  %% prefer including fewer literals
#show selected_literal/1.
```

This encoding will find the minimal set of `selected_literal` by utilizing the `#heuristic` directive in *clingo* which will prioritize assigning `false` to `selected_literal`. See Appendix Example 2 for an example.

3.1.3 Worked Example: Sufficient Explanations from a Decision Tree

Consider the decision tree in Figure 1 with instance $\boldsymbol{x} = \{x_1 = 1, x_2 = 1, x_3 = 2\}$.

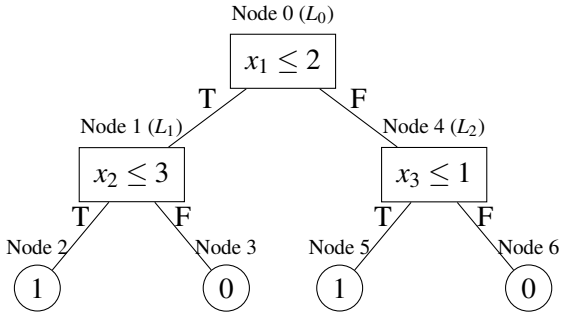


Figure 1: Example decision tree for Boolean attributes.

The preprocessing converts this tree to Boolean literals: $(x_1 \leq 2)$ becomes literal L_0 (true), $(x_2 \leq 3)$ becomes literal L_1 (true), and $(x_3 \leq 1)$ becomes literal L_2 (false). The instance follows the left-left path, reaching leaf node 2 with prediction 1. The ASP encoding represents this instance as a set of facts:

```
pre_class(1). node(0,0,1). node(1,1,1). node(4,2,0).
leaf_node(2,1). leaf_node(3,0). leaf_node(5,1). leaf_node(6,0).
left_node(0,1). right_node(0,4). % ... additional facts
```

For sufficient explanation, Program 1 produces `selected_literal(2)`, meaning literals L_0 and L_1 form the sufficient explanation $\{x_1 \leq 2, x_2 \leq 3\}$.

3.2 Random Forest

Random forests are ensembles of decision trees where the final prediction is determined by majority voting. We describe how sufficient, contrastive, and majority explanations can be derived from random forests using ASP. Each tree is encoded as described previously, with additional predicates to track selected literals and tree agreement. Table 3 lists the predicates used to represent forest structure and control explanation logic.

3.2.1 Sufficient Explanation from a Random Forest

Definition 3 (Sufficient Explanation [3]) *Let $RF = \{T_1, \ldots, T_m\}$ be a random forest over Boolean variables $X_n = \{x_1, \ldots, x_n\}$, and let f be the Boolean function represented by RF. Let $\boldsymbol{x} \in \{0,1\}^n$ be an instance with associated term $t_{\boldsymbol{x}}$. A sufficient explanation for $\boldsymbol{x}$ to be classified 1 (resp. 0) given f is a subterm t of $t_{\boldsymbol{x}}$ such that t is a prime implicant of f (resp. $\neg f$).*

Predicate	Meaning
`node(T,X,L,B)`	In tree `T`, node `X` checks whether literal `L` has truth value `B` for this instance.
`class(T,C)`	The tree `T` predicts class `C` for this instance.
`feature(L,B)`	The literal `L` has truth value `B` for this instance.
`new_node(T,X,L,B)`	Copy of the node with truth value possibly flipped.
`next_node(T,X)`	Reachable node under possibly changed truth value.
`leaf_node(T,X,C)`	In tree `T`, node `X` is a leaf predicting class `C`.
`left_node(T,X,LX)`	Left child of node `X` in tree `T` is `LX`.
`right_node(T,X,RX)`	Right child of node `X` in tree `T` is `RX`.
`pre_forest(P)`	Predicted class of the forest for $\boldsymbol{x}$ is `P`.
`tree_threshold(TH)`	Threshold for majority in sufficient explanations.
`con_tree_threshold(TH)`	Threshold for majority in contrastive explanations.
`majo_tree_threshold(TH)`	Threshold for majority explanations (e.g., $\lceil m/2 \rceil$).

Table 3: Predicates used to encode random forest models as ASP facts.

A *minimal unsatisfiable subset* (MUS) is a set of atoms that enforce the absence of an answer set. For random forests, sufficient explanations require fixing a minimal set of atoms that block all counterfactuals. We adopt a two-stage approach using *WASP* [1] with unsatisfiable core extraction: Stage 1 searches for counterfactuals that change the forest's prediction; Stage 2 computes a MUS of literals that must be fixed to prevent this change.

The first encoding below defines the logical structure of the forest and detects whether a change in the instance leads to a different prediction. The second encoding uses WASP's `--mus` mode to extract a minimal set of fixed literals. The following encoding computes sufficient explanations by identifying a maximal set of literals such that enough trees preserve the prediction.

Program 3: Sufficient Explanation from a Random Forest stage 1

```
1  node(T,X,L) :- node(T,X,L,_).
2  new_node(T,X,L,0) :- change(L),node(T,X,L),feature(L,1).
3  new_node(T,X,L,1) :- change(L),node(T,X,L),feature(L,0).
4  new_node(T,X,L,B) :- node(T,X,L),feature(L,B),not change(L).
5  next_node(T,LX) :- new_node(T,0,L,1),left_node(T,0,LX).
6  next_node(T,RX) :- new_node(T,0,L,0),right_node(T,0,RX).
7  next_node(T,LX) :- next_node(T,X),new_node(T,X,L,1),left_node(T,X,LX).
8  next_node(T,RX) :- next_node(T,X),new_node(T,X,L,0),right_node(T,X,RX).
9  class(T,C) :- next_node(T,X),leaf_node(T,X,C).
10 invalid_tree(T) :- class(T,C),pre_forest(FC),C!=FC.
11 forest_changed :- VT=#count{T:invalid_tree(T)},tree_threshold(TH),VT>TH.
12 :- not forest_changed.
```

The second stage computes a MUS of literals that must be fixed to prevent the forest's prediction from changing, based on the counterfactual structure defined in Stage 1. This is done by invoking WASP with `--mus` mode, which extracts a minimal unsatisfiable core over the guessed subset of fixed literals.

Program 4: Sufficient Explanation from a Random Forest stage 2

```
1  feature(I,B). ...  %% list of features involved in the prediction of forest
2  fix_lit(I). ...  %% guessing atoms for MUS extraction (see Appendix)
```

Program 4 shows the high-level structure. The `feature/2` facts represent the original instance, while `fix_lit/1` guesses which literals must retain their original value. This method allows us to extract a

minimal set of literals that are *necessary* to preserve the original prediction, effectively computing a sufficient explanation. See Appendix Example 3 for an example.

3.2.2 Contrastive Explanation from a Random Forest

Definition 4 (Contrastive Explanation [3]) *Let $RF = \{T_1, \ldots, T_m\}$ be a random forest over Boolean variables $X_n = \{x_1, \ldots, x_n\}$, let f be the Boolean function represented by RF, and let $\boldsymbol{x} \in \{0,1\}^n$ be an instance with associated term $t_{\boldsymbol{x}}$. A contrastive explanation for $\boldsymbol{x}$ to be classified 1 (resp. 0) given f is a subterm t of $t_{\boldsymbol{x}}$ such that removing the literals in t from $t_{\boldsymbol{x}}$ yields a term that is not an implicant of f (resp. $\neg f$), and for every literal $l \in t$, removing l from t does not satisfy this condition.*

This encoding searches for a minimal set of literals whose removal flips the random forest output. The `selected_literal/1` predicate determines which literals change, and `con_tree_threshold/1` enforces the required number of disagreeing trees.

Program 5: Contrastive Explanation from a Random Forest

```
1  1 {selected_literal(L):node(T,X,L,B)}.
2  new_node(T,X,L,0) :- selected_literal(L),node(T,X,L,1).
3  new_node(T,X,L,1) :- selected_literal(L),node(T,X,L,0).
4  new_node(T,X,L,B) :- node(T,X,L,B),not selected_literal(L).
5  next_node(T,LX) :- new_node(T,0,L,1),left_node(T,0,LX).
6  next_node(T,RX) :- new_node(T,0,L,0),right_node(T,0,RX).
7  next_node(T,LX) :- next_node(T,X),new_node(T,X,L,1),left_node(T,X,LX).
8  next_node(T,RX) :- next_node(T,X),new_node(T,X,L,0),right_node(T,X,RX).
9  class(T,C):-next_node(T,X),leaf_node(T,X,C).
10 valid_tree(T) :- class(T,C),pre_forest(FC),C!=FC.
11 valid :- VT = #count{T : valid_tree(T)},con_tree_threshold(TH),VT>TH.
12 :- not valid.
13 #heuristic selected_literal(L). [1,false]
14 #show selected_literal/1.
```

3.2.3 Majority Explanation from a Random Forest

Definition 5 (Majority Explanation [3]) *Let $RF = \{T_1, \ldots, T_m\}$ be a random forest over Boolean variables $X_n = \{x_1, \ldots, x_n\}$, let f be the Boolean function represented by RF and let $\boldsymbol{x} \in \{0,1\}^n$ be an instance with associated term $t_{\boldsymbol{x}}$. A majority explanation for $\boldsymbol{x}$ to be classified 1 (resp. 0) given f is a subterm t of $t_{\boldsymbol{x}}$ such that t is an implicant of at least $\lfloor m/2 \rfloor + 1$ trees T_i if $RF(\boldsymbol{x}) = 1$ (resp. $\neg T_i$ if $RF(\boldsymbol{x}) = 0$), and for every literal $l \in t$, removing l from t does not satisfy this condition.*

This encoding searches for a maximal term t that can be flipped without violating the majority vote by counting trees that agree with the forest's original prediction. The explanation must retain enough agreement across individual trees to ensure that the forest prediction remains consistent.

Program 6: Majority Explanation from a Random Forest

```
1  1 {selected_literal(L):node(T,X,L,B)}.
2  node(T,X,L,0..1) :- selected_literal(L),node(T,X,L,B).
3  next_node(T,LX) :- node(T,0,L,1),left_node(T,0,LX).
4  next_node(T,RX) :- node(T,0,L,0),right_node(T,0,RX).
5  next_node(T,LX) :- next_node(T,X),node(T,X,L,1),left_node(T,X,LX).
6  next_node(T,RX) :- next_node(T,X),node(T,X,L,0),right_node(T,X,RX).
7  class(T,C) :- next_node(T,X),leaf_node(T,X,C).
```

```
8  invalid_tree(T) :- class(T,C),pre_forest(FC),C!=FC.
9  valid :- VT = #count{T : invalid_tree(T)},majo_tree_threshold(TH),VT<TH.
10 :- not valid.
11 #heuristic selected_literal(L). [1,true]
12 #show selected_literal/1.
```

This encoding uses `majo_tree_threshold/1` to specify the minimum number of agreeing trees required to preserve the original forest prediction. The `#heuristic` directive encourages literal inclusion, producing a maximal term that is filtered to identify minimal majority explanations. Since the encoding finds maximal sets of literals that can be flipped, the actual majority explanation is the complement: the literals that must remain fixed to preserve the majority vote. See Appendix Example 5 for an example.

3.3 Boosted Trees

We now turn to boosted tree models, such as XGBoost [7] and LightGBM [17], where prediction is made by aggregating outputs from regression trees. Let $BT = \{T_1, \ldots, T_m\}$ $(1 \leq i \leq m)$ composed of m trees denote a boosted tree model, where each leaf contributes a real-valued weight and the final prediction is determined by the sign of the sum of weights. Table 4 lists the predicates used to encode boosted trees as ASP facts.

Predicate	Meaning
`node(T,X,L,B)`	In tree, `T`, node `X` checks whether literal `L` has truth value `B` in the current instance.
`leaf_node(T,X,W)`	Node `X` in tree `T` is a leaf with weight `W`.
`left_node(T,X,LX)`	Left child of node `X` in tree `T` is `LX`.
`right_node(T,X,RX)`	Right child of node `X` in tree `T` is `RX`.
`pre_forest(P)`	Original prediction of `X` by the forest is `P` $\in \{0,1\}$.
`weight(T,W)`	Leaf weight reached under current selection in tree `T`.
`best_weight(T,BW)`	Maximum weight reachable in tree `T`.
`worst_weight(T,WW)`	Minimum weight reachable in tree `T`.

Table 4: Predicates used to encode boosted tree models as ASP facts.

3.3.1 Tree-specific explanation from a Boosted Tree

Tree-specific explanations for boosted trees identify minimal sets of literals that guarantee a prediction under best-case or worst-case scenarios.

Definition 6 (Best and Worst Instance [4]) *Let $BT = \{T_1, \ldots, T_m\}$ be a boosted tree over Boolean variables $X_n = \{x_1, \ldots, x_n\}$, and let $\boldsymbol{x}, \boldsymbol{x}', \boldsymbol{x}'' \in \{0,1\}^n$ be instances with corresponding terms $t_{\boldsymbol{x}}, t_{\boldsymbol{x}'}, t_{\boldsymbol{x}''}$, and assume $BT(\boldsymbol{x}) = BT(\boldsymbol{x}') = BT(\boldsymbol{x}'')$.*

- *The best instance extending t is any $\boldsymbol{x}'$ such that t is a subterm of $t_{\boldsymbol{x}'}$, $BT(\boldsymbol{x}') = BT(\boldsymbol{x})$, and $\boldsymbol{x}' = \arg\max_{\boldsymbol{x}'':t \text{ is a subterm of } t_{\boldsymbol{x}''}}(\{w(BT, \boldsymbol{x}'')\})$. $w_{\uparrow}(t, BT)$ is the weight of any best instance, i.e., $w_{\uparrow}(t, BT) = \max_{\boldsymbol{x}'':t \text{ is a subterm of } t_{\boldsymbol{x}''}} w(BT, \boldsymbol{x}'')$.*
- *The worst instance extending t is any $\boldsymbol{x}'$ such that t is a subterm of $t_{\boldsymbol{x}'}$, $BT(\boldsymbol{x}') = BT(\boldsymbol{x})$, and $\boldsymbol{x}' = \arg\min_{\boldsymbol{x}'':t \text{ is a subterm of } t_{\boldsymbol{x}''}}(\{w(BT, \boldsymbol{x}'')\})$. $w_{\downarrow}(t, BT)$ is the weight of any worst instance, i.e., $w_{\downarrow}(t, BT) = \min_{\boldsymbol{x}'':t \text{ is a subterm of } t_{\boldsymbol{x}''}} w(BT, \boldsymbol{x}'')$.*

The best (resp. worst) instances represent the most (resp. least) favorable scenarios, allowing reasoning about prediction guarantees under pessimistic conditions.

Definition 7 (Tree-Specific Explanation [4]) *Let $BT = \{T_1, \ldots, T_m\}$ ($1 \leq i \leq m$) be a boosted tree, $\boldsymbol{x}$ be the input instance. Formally, let $w_{\downarrow}(t, T_i)$ denote the minimum (worst-case) score that tree T_i can produce under any instance extending t, and $w_{\uparrow}(t, T_i)$ the maximum (best-case) score. Then, t is a tree-specific explanation of instance $\boldsymbol{x}$ if it satisfies:*

- *When $BT(\boldsymbol{x}) = 1$: t is a subterm of $t_{\boldsymbol{x}}$, and $\sum_{i=1}^{m} w_{\downarrow}(t, T_i) > 0$ and no proper subterm of t satisfies the latter condition.*
- *When $BT(\boldsymbol{x}) = 0$: t is a subterm of $t_{\boldsymbol{x}}$, and $\sum_{i=1}^{m} w_{\uparrow}(t, T_i) \leq 0$ and no proper subterm of t satisfies the latter condition.*

Program 7: Tree-specific Explanation from a Boosted Tree

```
1  1 {selected_literal(L) : node(T,X,L,B) }.
2  node(T,X,L,0..1) :- node(T,X,L,B), not selected_literal(L).
3  next_node(T,LX) :- node(T,0,L,1), left_node(T,0,LX).
4  next_node(T,RX) :- node(T,0,L,0), right_node(T,0,RX).
5  next_node(T,LX) :- next_node(T,X), node(T,X,L,1), left_node(T,X,LX).
6  next_node(T,RX) :- next_node(T,X), node(T,X,L,0), right_node(T,X,RX).
7  weight(T,W) :- next_node(T,X), leaf_node(T,X,W).
8  best_weight(T,BW) :- weight(T, _), BW = #max{W:weight(T,W)}.
9  worst_weight(T,WW) :- weight(T, _), WW = #min{W:weight(T,W)}.
10 valid :- SW = #sum{BW:best_weight(_,BW)}, SW<=0, pre_forest(0).
11 valid :- SW = #sum{WW:worst_weight(_,WW)}, SW>0, pre_forest(1).
12 :- not valid.
13 #heuristic selected_literal(L). [1,false]
14 #show selected_literal/1.
```

This encoding preserves decisions under pessimistic aggregation using `#min` and `#max` aggregates (Lines 8 and 9) for modelling tree extremal behavior and `#sum` aggregates (Lines 10 and 11) for forest-level decisions. See Appendix Example 6 for an example.

4 Experiments

To evaluate our approach, we conducted experiments on 20 publicly available datasets from OpenML, UCI, and Kaggle. We used 100 test instances per dataset sampled from a 20% test split, to ensure that evaluation reflects generalization performance on unseen data. We set a 100-second timeout per instance, a maximum depth of 6, and 100 estimators for ensemble models.

We compare our ASP-based method against PyXAI [2], a SAT-based framework for formal explanations. LIME [21], SHAP [18] and Anchors [22] are excluded from this comparison, since they provide heuristic approximations without formal guarantees [13]. Experiments were conducted on Ubuntu 22.04 with Intel Core i9-9900K and 64 GB RAM.

Table 5 shows success rates for computing explanations within the timeout, distinguishing between computing *all* explanations versus *one* explanation. A successful computation means that at least one explanation was produced for the *one* case, and that all explanations were enumerated for the *all* case. Our ASP approach achieved perfect success on decision trees and was competitive on random forests when computing a single explanation. However, ASP struggled when enumerating explanations for RFs, especially for contrastive and majority explanations. In contrast, for enumerating explanations for RFs, PyXAI showed better performance in enumerating contrastive explanations, but ASP was better in enumerating majority explanations.

Type	Num. explanations	Model	PyXAI completed (%)	ASP completed (%)
Contrastive	All	DT	100.0	100.0
Contrastive	All	RF	42.6	6.4
Contrastive	One	DT	100.0	100.0
Contrastive	One	RF	60.8	100.0
Majority	All	RF	0.4	6.4
Majority	One	RF	100.0	98.7
Sufficient	All	DT	100.0	100.0
Sufficient	One	DT	100.0	100.0
Sufficient	One	RF	100.0	57.7
Tree Specific	One	XGB	100.0	20.8

Table 5: Success rates of computing explanations across explanation types and models.

Table 6 compares average number of explanations and lengths for Random Forest models, showing results only for completed instances. Note that averages are computed only over successful instances, and PyXAI and ASP methods may succeed on different subsets.

ASP was able to enumerate more contrastive and majority explanations than PyXAI on the subset of instances where it completed execution within the timeout (#N columns). This trend is particularly evident in the case of majority explanations, where ASP outperformed PyXAI significantly on several datasets (e.g., *startup*, *titanic*, *breast*), despite a lower overall completion rate. However, ASP faces scalability challenges for complex models and exhaustive enumeration under resource constraints.

5 Related Work

The problem of explaining decision tree-based models has received significant attention, particularly through logical and symbolic methods. Audemard et al. [2] proposed a SAT-based framework for generating formal explanations from decision trees by translating the model into CNF, supporting *direct*, *sufficient*, and *contrastive* that correspond to the explanation notions we adopt in this paper.

Izza et al. [15] addressed redundancy in path-based explanations by extracting abductive explanations as literal subsets, offering more concise explanations than path tracing.

Several SAT-based extensions have targeted ensemble models. Audemard et al. extended their framework to compute *majority explanations* [3] and *contrastive explanations* [5] for random forests, demonstrating feasibility while acknowledging computational challenges in explanation enumeration.

Izza et al. [16] proposed an efficient SAT-based method to generate prime implicant (PI) explanations for random forests. Their results showed improved performance over heuristic methods such as Anchors [22], highlighting the advantages of logical methods in producing faithful explanations.

Takemura et al. [24] proposed an ASP-based method to derive rule-based explanations from decision tree models. Their approach focused on extracting global and local rules through pattern mining from the structure of decision and ensemble trees. However, their explanations lack the logical guarantees provided by our approach.

Beyond tree models, Dhurandhar et al. [8] developed the Contrastive Explanation Method (CEM) for deep neural networks, introducing the contrastive paradigm and emphasizing explanations aligned with human intuition through the presence or absence of signals.

In contrast to these works, our ASP-based approach provides a unified, extensible framework for generating multiple types of formal explanations for decision tree models and ensembles. By leveraging

	dataset	Con #N		Maj #N		Con #L		Maj #L	
		PyXAI	ASP	PyXAI	ASP	PyXAI	ASP	PyXAI	ASP
1	ad_data	19.7	-	-	133277.5	1.7	-	-	181.1
2	adult	11.8	-	-	72112.0	2.2	-	-	77.3
3	arcene	179.0	179.0	-	-	1.0	1.0	-	-
4	australia	7.9	-	-	-	1.8	-	-	-
5	banknote	8.0	31.0	-	-	1.7	1.0	-	-
6	breast	8.3	10.0	27.7	48202.8	2.7	1.0	33.8	30.6
7	christine	2.7	157.0	-	-	2.4	1.0	-	-
8	compas	3.9	3.8	-	22085.6	1.5	1.5	-	13.9
9	credit_german	7.5	-	-	1392.0	2.6	-	-	82.6
10	dexter	50.7	172.8	-	-	2.2	1.0	-	-
11	diabetes	11.7	-	-	-	1.7	-	-	-
12	divorce	20.6	-	-	-	4.4	-	-	-
13	gina_agnostic	7.2	-	-	-	2.3	-	-	-
14	heart	6.7	-	-	5768.0	2.0	-	-	81.1
15	indian_liver_patient	24.5	-	-	-	1.5	-	-	-
16	qsar	9.9	-	-	-	2.0	-	-	-
17	spambase	1.7	-	-	-	1.2	-	-	-
18	startup	14.0	476.1	-	-	2.1	1.0	-	-
19	titanic	13.9	208.0	-	127778.0	2.5	1.0	-	212.9
20	wine	6.0	-	-	-	1.0	-	-	-

Table 6: Random Forest results: average number (#N) and length (#L) of contrastive and majority explanations. Dash (-) indicates timeouts (100s). Results for completed instances only (all explanations).

ASP's model enumeration capabilities, we enable various explanation types within a single encoding environment, improving flexibility and simplifying integration with domain-specific constraints.

6 Conclusion

This work presented an ASP-based approach for generating formal explanations for decision tree models. We implemented encodings for sufficient, contrastive, majority, and tree-specific explanations, demonstrating that ASP can reproduce and sometimes outperform SAT-based methods, particularly for individual majority and contrastive explanations on random forests.

ASP's enumeration capabilities offer practical advantages when generating multiple explanations by exploring the search space in a single solver call. However, scalability challenges remain for large models, where long computation time limits practical applicability. The primary bottlenecks are: (1) exponential memory growth during grounding when instantiating all possible combinations, and (2) the inefficient two-stage process requiring two distinct ASP systems.

Future work will focus on improving runtime performance through more efficient encodings, pruning, better heuristics, and optimization, while extending to boosted tree explanations and broader comparative analysis.

Acknowledgements

This work has been supported by JSPS KAKENHI Grant Numbers JP21H04905, JP25K03190 and JST CREST Grant Number JPMJCR22D3, Japan.

References

[1] Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone & Francesco Ricca (2013): *WASP: A Native ASP Solver Based on Constraint Learning*. In: *LPNMR*, pp. 54–66, doi:10.1007/978-3-642-40564-8_6.

[2] Gilles Audemard, Steve Bellart, Louenas Bounia, Frederic Koriche, Jean-Marie Lagniez & Pierre Marquis (2022): *On the Explanatory Power of Boolean Decision Trees*. *Data and Knowledge Engineering* 142, p. 102088, doi:10.1016/j.datak.2022.102088.

[3] Gilles Audemard, Steve Bellart, Louenas Bounia, Frederic Koriche, Jean-Marie Lagniez & Pierre Marquis (2022): *Trading Complexity for Sparsity in Random Forest Explanations*. In: *AAAI*, 36, pp. 5461–5469, doi:10.1609/aaai.v36i5.20484.

[4] Gilles Audemard, Jean-Marie Lagniez, Pierre Marquis & Nicolas Szczepanski (2023): *Computing Abductive Explanations for Boosted Trees*. In: *AISTATS*, pp. 4699–4711.

[5] Gilles Audemard, Jean-Marie Lagniez, Pierre Marquis & Nicolas Szczepanski (2023): *On Contrastive Explanations for Tree-Based Classifiers*. In: *ECAI, Frontiers in Artificial Intelligence and Applications* 372, pp. 117–124, doi:10.3233/faia230261.

[6] Leo Breiman (2001): *Random Forests*. *Machine Learning* 45(1), pp. 5–32, doi:10.1023/A:1010933404324.

[7] Tianqi Chen & Carlos Guestrin (2016): *XGBoost: A Scalable Tree Boosting System*. In: *KDD*, pp. 785–794, doi:10.1145/2939672.2939785.

[8] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Pai-Shun Ting, Karthikeyan Shanmugam & Payel Das (2018): *Explanations Based on the Missing: Towards Contrastive Explanations with Pertinent Negatives*. In: *NeurIPS*, pp. 590–601.

[9] Jerome H. Friedman (2001): *Greedy Function Approximation: A Gradient Boosting Machine*. *Annals of statistics*, pp. 1189–1232, doi:10.1214/aos/1013203451.

[10] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2014): *Clingo = ASP + Control: Preliminary Report*. *CoRR* abs/1405.3694.

[11] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In: *ICLP/SLP*, 88, pp. 1070–1080.

[12] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti & Dino Pedreschi (2018): *A Survey of Methods for Explaining Black Box Models*. *ACM Computing Surveys* 51(5), p. Article 93, doi:10.1145/3236009.

[13] Alexey Ignatiev (2020): *Towards Trustable Explainable AI*. In: *IJCAI*, 5, pp. 5154–5158, doi:10.24963/ijcai.2020/726.

[14] Alexey Ignatiev, Nina Narodytska & João Marques-Silva (2019): *Abduction-Based Explanations for Machine Learning Models*. In: *AAAI*, pp. 1511–1519, doi:10.1609/aaai.v33i01.33011511.

[15] Yacine Izza, Alexey Ignatiev & João Marques-Silva (2020): *On Explaining Decision Trees*. arXiv:2010.11034.

[16] Yacine Izza & João Marques-Silva (2021): *On Explaining Random Forests with SAT*. In: *IJCAI*, pp. 2584–2591, doi:10.24963/ijcai.2021/356.

[17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye & Tie-Yan Liu (2017): *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*. In: *NeurIPS*, pp. 3146–3154.

[18] Scott M Lundberg & Su-In Lee (2017): *A Unified Approach to Interpreting Model Predictions*. In: *NeurIPS*, pp. 4765–4774.

[19] João Marques-Silva & Alexey Ignatiev (2022): *Delivering Trustworthy AI through Formal XAI*. In: *AAAI*, pp. 12342–12350, doi:10.1609/aaai.v36i11.21499.

[20] Nina Narodytska, Aditya Shrotri, Kuldeep S. Meel, Alexey Ignatiev & João Marques-Silva (2019): *Assessing Heuristic Machine Learning Explanations with Model Counting*. In: *SAT*, pp. 267–278, doi:10.1007/978-3-030-24258-9_19.

[21] Marco Tulio Ribeiro, Sameer Singh & Carlos Guestrin (2016): *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. In: *KDD*, pp. 1135–1144, doi:10.1145/2939672.2939778.

[22] Marco Tulio Ribeiro, Sameer Singh & Carlos Guestrin (2018): *Anchors: High-precision Model-Agnostic Explanations*. In: *AAAI*, pp. 1527–1535, doi:10.1609/aaai.v32i1.11491.

[23] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova & Chudi Zhong (2022): *Interpretable Machine Learning: Fundamental Principles and 10 Grand Challenges*. *Statistics Surveys* 16, pp. 1–85, doi:10.1214/21-ss133.

[24] Akihiro Takemura & Katsumi Inoue (2024): *Generating Global and Local Explanations for Tree-Ensemble Learning Methods by Answer Set Programming*. *Theory and Practice of Logic Programming* 24(5), pp. 973–1010, doi:10.1017/s1471068424000401.

A Illustrative Examples of Explanations

This appendix provides visual examples to illustrate how sufficient, contrastive, majority and tree-specific explanations can be derived from decision tree models. Each figure is accompanied by a brief explanation of what the model structure and literals represent.

A.1 Decision Tree

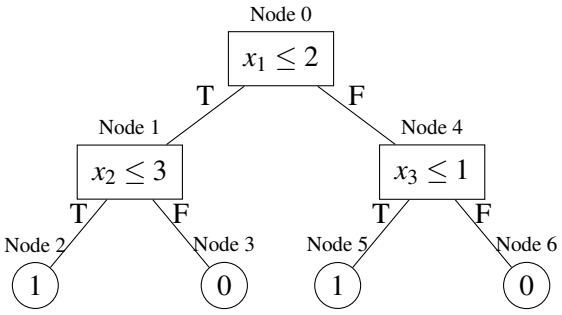


Figure 2: A decision tree with three internal nodes.

Let the input instance be $\boldsymbol{x} = \{x_1 = 1, x_2 = 1, x_3 = 2\}$. This instance satisfies $(x_1 \leq 2)$ and $(x_2 \leq 3)$, so the tree follows the left-left path and predicts 1. The tree is encoded into a set of ASP facts, where we use the depth-first indexing starting at 0. For example, `node(0,1,0)` corresponds to $(x_1 \leq 2)$ being satisfied, and `node(1,1,1)` corresponds to $(x_2 \leq 3)$ being satisfied, etc.

Program 8: Decision Tree: Instance

```
%% node status
node(0,0,1). node(1,1,1). node(4,1,0).

%% tree structure
leaf_node(2,1). leaf_node(3,0). leaf_node(5,1). leaf_node(6,0).
left_node(0,1). right_node(0,4). left_node(1,2). right_node(1,3).
left_node(4,5). right_node(4,6).
```

Example 1 (Sufficient Explanation) *Executing the Program 1 with this instance produces the answer set containing `selected_literal(2)`, which is the complement of the sufficient explanation. Thus, the sufficient explanation for this case is: `selected_literal(0)` and `selected_literal(1)`. These correspond to the literals $\{x_1 \leq 2, x_2 \leq 3\}$, where changing either literal causes the prediction to flip, making the term a sufficient explanation.*

Example 2 (Contrastive Explanation) *Executing the program 2 yields `selected_literal(1)`, which corresponds to $\{x_2 \leq 3\}$. Flipping this condition, i.e., $x_2 > 3$, causes the output to change from 1 to 0: this is a contrastive explanation.*

A.2 Random Forest

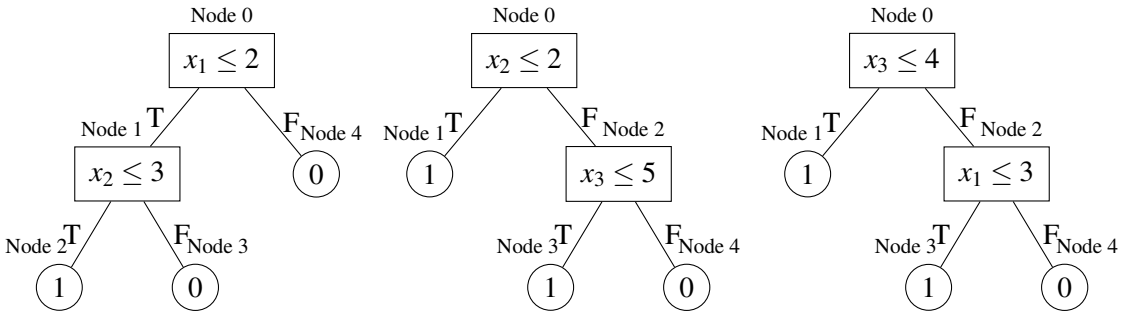


Figure 3: A random forest with three trees.

Let us consider the instance $\boldsymbol{x} = \{x_1 = 1, x_2 = 2, x_3 = 6\}$, for which all three trees predict class 1. This random forest and the instance results in the following encoding.

Program 9: Random Forest: Instance

```
%% instance and node status
feature(1,1).
feature(2,1).
feature(3,1).
feature(4,0).
feature(5,0).
feature(6,1).

%% forest prediction and thresholds
pre_forest(1). con_tree_threshold(1). majo_tree_threshold(2).

%% tree 1
node(1,0,1,1). node(1,1,2,1).
leaf_node(1,2,1). leaf_node(1,3,0). leaf_node(1,4,0).
left_node(1,0,1). right_node(1,0,4). left_node(1,1,2). right_node(1,1,3).

%% tree 2
node(2,0,3,1). node(2,2,4,0).
leaf_node(2,1,1). leaf_node(2,3,1). leaf_node(2,4,0).
left_node(2,0,1). right_node(2,0,2). left_node(2,2,3). right_node(2,2,4).

%% tree 3
node(3,0,5,0). node(3,2,6,1).
leaf_node(3,1,1). leaf_node(3,3,1). leaf_node(3,4,0).
left_node(3,0,1). right_node(3,0,2). left_node(3,2,3). right_node(3,2,4).
```

Program 10: Random Forest Sufficient Explanation: Encoding

```

%% Prepend this line to Program 3 and 9, and save it as enc.lp
%% Generator for MUS extraction. Note that 1..6 is forest-dependent.
{change(L) : not fix_lit(L),L=1..6}.

%% Program 3
node(T,X,L) :- node(T,X,L,_).
```

```
new_node(T,X,L,0) :- change(L),node(T,X,L),feature(L,1).
new_node(T,X,L,1) :- change(L),node(T,X,L),feature(L,0).
...

%% Program 9
%% instance and node status
feature(1,1).
feature(2,1).
...
```

Program 11: Random Forest Sufficient Explanation: Instance

```
%% For MUS extraction with WASP, save this as ins.lp
fix_lit(1).
fix_lit(2).
fix_lit(3).
fix_lit(4).
fix_lit(5).
fix_lit(6).
```

Program 12: Random Forest Sufficient Explanation: Command

```
python wasp_rewriter.py enc.lp ins.lp | clingo --output=smodels | ./wasp --mus=
    __debug__ -n0
```

Example 3 (Sufficient Explanation) *Executing the command in Program 12 with WASP outputs* `fix_lit(3), fix_lit(6)`*, which corresponds to* $\{x_2 \leq 2, x_1 \leq 3\}$*. Fixing these literals ensures that enough trees preserve the original prediction.*

Example 4 (Contrastive Explanation) *Executing the program 5 outputs* `selected_literal(1), selected_literal(3)`*, which corresponds to* $\{x_1 \leq 2, x_2 \leq 2\}$*. If both of these literals are flipped, the majority of the trees change their prediction, and the forest's prediction will change.*

Example 5 (Majority Explanation) *Executing the program 6 outputs* `selected_literal(3), selected_literal(4), selected_literal(5),` *which, when subtracted from the set of literals, results in* `selected_literal(1), selected_literal(2), selected_literal(6)`*. These correspond to* $\{x_1 \leq 2, x_2 \leq 3, x_1 \leq 3\}$*, which ensure that at least two trees maintain the prediction.*

A.3 Boosted Trees

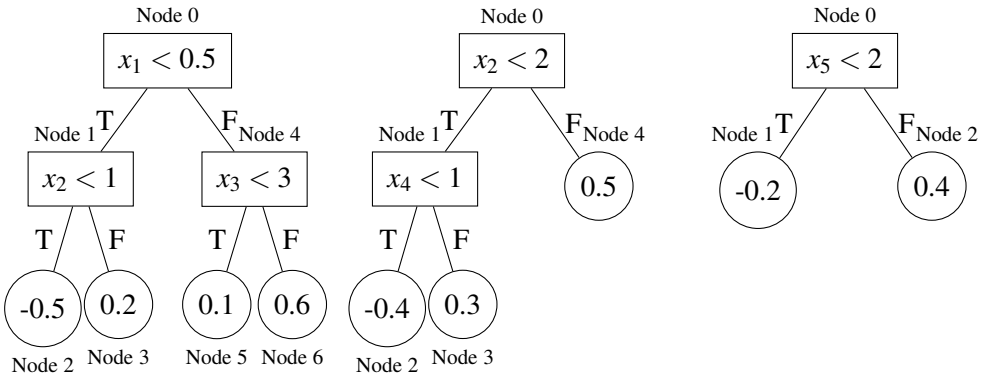


Figure 4: Boosted tree model with three regression trees.

Consider the input instance $\boldsymbol{x} = \{x_1 = 1,\ x_2 = 2,\ x_3 = 3,\ x_4 = 4,\ x_5 = 5\}$. The boosted tree is encoded into ASP facts as follows:

Program 13: Boosted Tree: Instance

```
pre_forest(1).
%% tree 1
node(0,0,1,0). node(0,1,2,1). node(0,4,3,0).
leaf_node(0,2,-500). leaf_node(0,3,200). leaf_node(0,5,100). leaf_node(0,6,600).
left_node(0,0,1). right_node(0,0,4).
left_node(0,1,2). right_node(0,1,3).
left_node(0,4,5). right_node(0,4,6).

%% tree 2
node(1,0,4,0). node(1,1,5,0).
leaf_node(1,2,-400). leaf_node(1,3,300). leaf_node(1,4,500).
left_node(1,0,1). right_node(1,0,4).
left_node(1,1,2). right_node(1,1,3).

%% tree 3
node(2,0,6,0). leaf_node(2,1,-200). leaf_node(2,2,400).
left_node(2,0,1). right_node(2,0,2).
```

Example 6 (Tree-Specific Explanation) *Executing the program 7 outputs* `selected_literal(5)`, `selected_literal(6)`, *which corresponds to* $\{x_4 < 1, x_5 < 2\}$. *Fixing these literals ensure that the sum of the worst-case instances remain greater than 0.*

xDNN$^{(ASP)}$: Explanation Generation System for Deep Neural Networks powered by Answer Set Programming

Ly Ly Trieu Tran Cao Son
New Mexico State University,
New Mexico, USA
lytrieu@nmsu.edu tson@cs.nmsu.edu

Explainable artificial intelligence (xAI) has gained significant attention in recent years. Among other things, explainablility for deep neural networks has been a topic of intensive research due to the meteoric rise in prominence of deep neural networks and their "black-box" nature. xAI approaches can be characterized along different dimensions such as their scope (global versus local explanations) or underlying methodologies (statistic-based versus rule-based strategies). Methods generating global explanations aim to provide reasoning process applicable to all possible output classes while local explanation methods focus only on a single, specific class. SHAP (SHapley Additive exPlanations), a well-known statistical technique, identifies important features of a network. Deep neural network rule extraction method constructs IF-THEN rules that link input conditions to a class. Another approach focuses on generating counterfactuals which help explain how small changes to an input can affect the model's predictions. However, these techniques primarily focus on the input-output relationship and thus neglect the structure of the network in explanation generation.

In this work, we propose xDNN$^{(ASP)}$, an explanation generation system for deep neural networks that provides global explanations. Given a neural network model and its training data, xDNN$^{(ASP)}$ extracts a logic program under answer set semantics that—in the ideal case—represents the trained model, i.e., answer sets of the extracted program correspond one-to-one to input-output pairs of the network. We demonstrate experimentally, using two synthetic datasets, that not only the extracted logic program maintains a high-level of accuracy in the prediction task, but it also provides valuable information for the understanding of the model such as the importance of features as well as the impact of hidden nodes on the prediction. The latter can be used as a guide for reducing the number of nodes used in hidden layers, i.e., providing a means for optimizing the network.

1 Introduction

Deep neural networks (DNN) have demonstrated their capability to successfully address complex problems (see, e.g., [4, 25]). However, the lack of transparency or the absence of an adequate explanation of the results can lead to users' mistrust or skepticism. A major problem, referred as *the problem model/algorithm explainability*, is concerned with users' struggle to understand why they receive specific recommendations and/or classification results. It is particularly concerning in critical domains such as healthcare and cyber-security (see, e.g., [4, 1]). The significance of explainability has increased with the introduce of "the right to an explanation" law by The European Parliament (see, e.g., [10, 15]). Thus, explainability has become a crucial and essential aspect in the proliferation of deep learning models.

The study in [15] classifies xAI methods into *local* and *global* explanations. Local explanation refers to the ability of explaining a specific prediction, while global explanation pertains to the overall model, encompassing all predictions. A well-know method for local explanations is Local Interpretable Model-Agnostic Explanations (LIME) ([7]). On the other hand, SHapley Additive exPlanation, SHAP ([18]),

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 438–452, doi:10.4204/EPTCS.439.30

© L.L. Trieu & T.C. Son
This work is licensed under the
Creative Commons Attribution License.

is a model-agnostic, statistically-based method that uses Shapley values to explain feature importance globally.

Rule extraction methods have also been developed for understanding DNNs. Some methods focus on generating counterfactuals which help identifying features that need to be changed to produce a different output ([14]). These methods allow users to understand how small changes in input can impact the model's decision. Some other approaches construct IF-THEN rules from the trained DNN model which are then used to explain the dependence between input feature values and outputs. This provides users with the understanding of the model behavior across all inputs. The work in [11] is a well-known method that proposes a procedure for inducing logic-based theory for a given neural model (referred to as the main network). The method has to utilize multiple mapping networks that are trained to map the activation values of the main network's output to human-defined concepts. The primary cost of this approach lies in the need to label data for the mapping network, which usually requires domain expert knowledge. Following a taxonomy proposed by [1] and [3] , rule extraction approaches can be divided into three categories: decompositional, pedagogical and eclectic. Decompositional methods derive rules based on the structure of the trained neural network and take into consideration hidden layers during extraction. On the other hand, pedagogical methods disregard the hierarchical structure in the extraction process. Eclectic methods integrate both decompositional and pedagogical approaches.

It is worth noticing that some feature/rules extraction methods directly from data ([28, 19] such as linear discriminant analysis (LDA) and decision tree could also be used to identify the importance of features and interpretable results. However, these methods do not explain the behavior of the given DNN and have difficulty dealing with irrelevant features.

In this paper, given a DNN model and an output, we are interested in the following questions: (1) "why is the output produced?", (2) "what features were involved?", and (3) "how was the interaction among the hidden layers?".

To answer the above questions, we propose an $\texttt{xDNN}^{\texttt{(ASP)}}$ system that belongs to the decompositional xAI group. Different from other existing methods such as DeepRED ([30]) and ECLAIRE ([29]), we take advantage of the answer set programming in producing the set of logic rules in order to provide explanation for answering the three questions. To the best of our knowledge, this is the first work to extract rules in answer set programming from a DNN model.

2 Background

In this work, we will translate DNNs into logic programs under the answer set semantics [12]. A logic program P is a set of rules of the form "$c \leftarrow a_1, \ldots, a_m, not\ b_1, \ldots, not\ b_n$" where c, a_i's, and b_j's are atoms of a first order language. We refer the readers to [12] for the precise definition of answer sets of logic programs[1].

Additionally, we utilize the decision tree classification algorithm in Waikato Environment for Knowledge Analysis (WEKA) as a baseline. WEKA is a well-known open-source software platform widely used for data mining and machine learning, developed at the University of Waikato in New Zealand [16]. It offers simple and user-friendly interfaces that make it easy to create fundamental machine learning models [23]. WEKA has been referenced widely (see, e.g. [5, 6, 24]). It has also been successfully applied in various domains, including health care and cybersecurity [27, 9].

[1] Answer sets can be computed using answer set solvers, such as *clingo* https://potassco.org

2.1 Deep Neural Network

A neural network consists of an input layer i, an output layer o, and multiple hidden layers $h_1,\ldots,h_j$, $j \geq 1$, between them ([13]). Each layer consists of a set of nodes, each node is associated with a bias and an activation function. Node j at layer $l-1$ provides an input with weight w_{ij}^l to node i at layer l. The matrices representing the weights and biases of the network are denoted by $\mathbf{w}$ and $\mathbf{b}$, respectively. The input layer i receives raw features from data, while the output layer o produces the neural network's prediction.

The number of nodes in each layer depends on the application and the designer of the network. Usually, the number of nodes in the input and output layer corresponds to the number of features in the input data and the number of classes of the classification task, respectively.

Given a training set[2] $\{(\mathbf{x_i}, y_i) \mid i = 1,...,N\}$, the goal of training a network is to learn $\mathbf{w}$ to minimize the error function $E(\mathbf{w},\mathbf{b}) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ where $\hat{y}_i$ represents the output value of the network given the input $\mathbf{x_i}$. a_i^l is the activation value at node i at layer l, whose activation function is f, is computed by $a_i^l = f(\sum_j w_{ij}^l a_j^{l-1} + b_i^l)$ (See, e.g., ([8]) for more details). Different activation functions can be used (e.g., *Tanh*, *Sigmoid*, or *ReLU*).

2.2 The Algorithm C5.0

The algorithm `C5.0` was introduced by [22] for a generating classification model from a set of tuples $\{(\mathbf{x_i}, c_i) \mid i = 1,...,N\}$ where $\mathbf{x_i}$ is the attribute values and c_i is its class. Compared to its predecessors, `C4.5` [21] and ID3 [20]), `C5.0` offers faster processing, better handling of large datasets, and more efficient memory usage. Detailed formulae for computing these values can be found in the books by [28] and [17]. For the dataset in Table 1, a set of eight tuples with two attributes *h_2_n_0* and *h_2_n_1* and the *xor* class, Algorithm `C5.0` selects *h_2_n_0* with the threshold **-0.37244028** and yields a decision tree that can be converted into the following two rules with $\mathbf{v} = -0.37244028$:

$$(r_1)\ IF\ h_2_n_0 \leq \mathbf{v}\ THEN\ xor = 0 \quad \text{and} \quad (r_2)\ IF\ h_2_n_0 > \mathbf{v}\ THEN\ xor = 1$$

Table 1: C5.0 example

h_2_n_0	h_2_n_1	xor
-0.4413446	0.6062831	0
-0.23081768	-0.6257931	1
0.5207858	0.9767507	1
-0.37244028	0.6900585	0
-0.4413446	0.6062831	0
-0.23081768	-0.6257931	1
0.5207858	0.9767507	1
-0.37244028	0.6900585	0

Given a set of rules returned by `C5.0`, a rule r's coverage and correctness, denoted by $r[\text{“}cover\text{”}]$ and $r[\text{“}ok\text{”}]$, respectively, are defined by the number of training instances that satisfy r's condition and that are correctly classified, respectively.

The confidence value of rule r is defined by $\frac{r[\text{“}ok\text{”}]}{r[\text{“}cover\text{”}]} * 100$. The THEN part contains the predicted class, denoted as $r[\text{“}class\text{”}]$. For example, the confidence value of r_1 wrt. the data in Table 1 is 1 and $r_1[\text{“}class\text{”}] = 0$. In the following, we often write $t = h_n_n_i \odot v$ to denote the condition between *IF* and *THEN* of a rule r, where $\odot \in \{\leq, >\}$ and use $t[\text{“}level\text{”}], t[\text{“}node\text{”}], t[\text{“}operation\text{”}]$ and $t[\text{“}threshold\text{”}]$ to refer to n, i, $\odot$, and v, respectively. For example, for $t = h_2_n_0 \leq -0.37244028$, $t[\text{“}level\text{”}]$, $t[\text{“}node\text{”}]$, $t[\text{“}operation\text{”}]$ and $t[\text{“}threshold\text{”}]$ are 2, 0, $\leq$, and -0.37244028, respectively.

[2]For simplicity of the representation, we assume a network with one node at the output layer. Multiple output nodes are formalized similarly.

3 xDNN$^{(ASP)}$ System

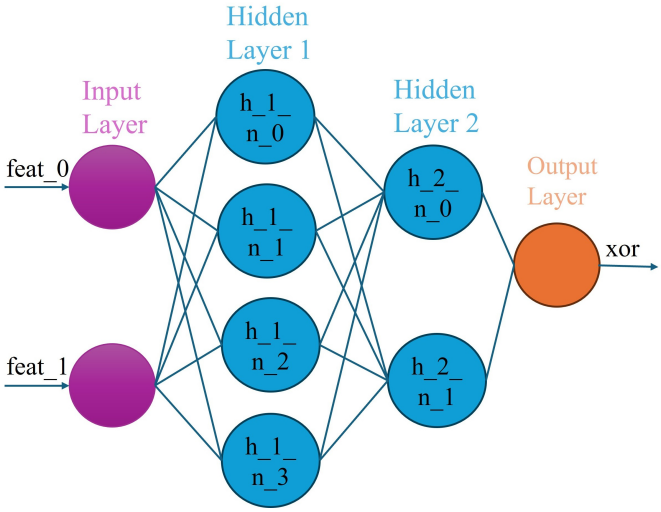


Figure 1: M_{xor}: An Example of a DNN.

This section describes the $xDNN^{(ASP)}$ system that generates a logic program representation $\Pi(M)$ of a DNN model M whose training data set is $\{(\mathbf{x_i}, y_i) \mid i \in \{1,2,...,N\}$. $\Pi(M)$ will be constructed using the network structure and the activation values at each nodes of M. Assume that M has k hidden layers. We write $h_i^j(\mathbf{x})$, $1 \leq i \leq k$, to denote the activation value at node j of layer i for the instance $\mathbf{x}$. We will begin with a discussion on the atoms and rules of $\Pi(M)$ and continue with the description of the algorithms. For illustration purpose, we train a DNN model, M_{xor} (Fig. 1), with two binary input features ($feat_0$ and $feat_1$), one output XOR, two hidden layers (4 nodes in the first hidden layer, 2 in the second), and the *tanh* activation function for all nodes. The training set consists of eight tuples, representing the table of the XOR gates, repeated two times[3] . It is done using 500 epochs, and a batch size of 4.

3.1 Atom and Rule Formats

Given a model M, atoms in $\Pi(M)$ correspond to the nodes in M. They are classified into three categories:

- *Input atom* is of the form $input(x,y)$ where x and y is the name and value of a feature;
- *Intermediate atom* represents condition associated with a node j in hidden layer i and takes one of the forms: $h(i,j,"\leq",v,true)$ or $h(i,j,">",v,true)$
  where v is the threshold associated to the rule that is generated by C5.0 and contains the condition related to node j at layer i, e.g., $h_2_n_0 \leq -0.37244028$ in rule r_1 in Section 2.2 is represented as $h(2,0,"\leq","-0.37244028",true)$.
- *Class atom* is encoded using predicate $potential_predict_output(x,y,z)$ where x denotes the predicted class label, y is the identifier of the rule, and z is a confidence value associated with the prediction in the rule.

Rules in $\Pi(M)$ encode the relationships between nodes. For a rule $head \leftarrow body$ in $\Pi(M)$, if the *head* is an atom corresponds to a node at level i of M, where $i = 0$ ($i = k+1$) represents the input (output) layer, then the *body* can only contain literals corresponding to nodes at level $i-1$. For ease of the

[3]This is a simplified version of the XOR dataset from https://www.kaggle.com/, an online platform for data scientists and machine learning professionals, that repeats the XOR table 2500 times.

presentation, rules are organized into three levels corresponding to the three types of connections in the neural network (input-to-hidden, hidden-to-hidden, and hidden-to-output), as follows: (*i*) *Top-level* rules have a class atom as the head and the intermediate literals from the last hidden layer as the body; (*ii*) *Intermediate rules* are those whose head and body contain only intermediate literals. (*iii*) *Bottom-level rules* have an intermediate literal from the first hidden layer as the head. To simplify the experiment, we construct a bottom-level rule with a body consists of (*i*) input atoms and (*ii*) arithmetic comparisons between the variable in the second term of an input atom and a threshold v derived from the trained model.

3.2 Algorithms of xDNN$^{(ASP)}$

Algorithm 1 *xDNN$^{(ASP)}$*

Input: a DNN model M with hidden layers $\{h_1,...,h_k\}$ and
training dataset $X = \{(\mathbf{x}^{(1)},y_1),(\mathbf{x}^{(2)},y_2),..,(\mathbf{x}^{(N)},y_N)\}$
Output: Answer set program $\Pi(M)$

1: $data_{top} \leftarrow \{(h_k(\mathbf{x}^{(1)}),y_1),(h_k(\mathbf{x}^{(2)}),y_2),...,(h_k(\mathbf{x}^{(N)}),y_N)\}$
2: $\Pi(M)_{top}, Conditions \leftarrow extract_top_level(data_{top})$
3: $\Pi(M) \leftarrow \Pi(M)_{top}$
4: **for** hidden layer $i = k-1,k-2,..,1$ **do**
5: $\quad tem_Conditions \leftarrow \emptyset$
6: $\quad$ **for** $t \in Conditions$ **do**
7: $\quad\quad h_{level}, n_{index}, \odot, th \leftarrow t[\text{“}level\text{”}], t[\text{“}node\text{”}], t[\text{“}operator\text{”}], t[\text{“}threshold\text{”}]$
8: $\quad\quad y'_1, y'_2, ..., y'_N \leftarrow t(h^{n_{index}}_{h_{level}}(\mathbf{x}^{(1)})), t(h^{n_{index}}_{h_{level}}(\mathbf{x}^{(2)})), .., t(h^{n_{index}}_{h_{level}}(\mathbf{x}^{(N)}))$
9: $\quad\quad data^t_i \leftarrow \{(h_i(\mathbf{x}^{(1)}),y'_1),(h_i(\mathbf{x}^{(2)}),y'_2),...,(h_i(\mathbf{x}^{(N)}),y'_N)\}$
10: $\quad\quad \Pi(M)^t_i, Conditions^t_i \leftarrow extract_intermediate_level(data^t_i, h_{level}, n_{index}, \odot, th)$
11: $\quad\quad \Pi(M) \leftarrow \Pi(M) \cup \Pi(M)^t_i$
12: $\quad\quad tem_Conditions \leftarrow tem_Conditions \cup Conditions^t_i$
13: $\quad Conditions \leftarrow tem_Conditions$
14: **for** $t \in Conditions$ **do**
15: $\quad 1, n_{index}, \odot, th \leftarrow t[\text{“}level\text{”}], t[\text{“}node\text{”}], t[\text{“}operator\text{”}], t[\text{“}threshold\text{”}]$
16: $\quad y'_1, y'_2, ..., y'_N \leftarrow t(h^{n_{index}}_1(\mathbf{x}^{(1)})), t(h^{n_{index}}_1(\mathbf{x}^{(2)})), .., t(h^{n_{index}}_1(\mathbf{x}^{(N)}))$
17: $\quad data^t_{input} \leftarrow \{(\mathbf{x}^{(1)},y'_1),(\mathbf{x}^{(2)},y'_2),...,(\mathbf{x}^{(N)},y'_N)\}$
18: $\quad \Pi(M)^t_{bottom} \leftarrow extract_bottom_level(data^t_{input}, 1, n_{index}, \odot, th)$
19: $\quad \Pi(M) \leftarrow \Pi(M) \cup \Pi(M)^t_{bottom}$
20: **return** $\Pi(M)$

Algorithm 1 is the main algorithm for constructing $\Pi(M)$ given M with k hidden layers and its training dataset $X = \{(\mathbf{x}^{(1)},y_1),(\mathbf{x}^{(2)},y_2),..,(\mathbf{x}^{(N)},y_N)\}$. It works as follows.

- *Computing top-level rules, $\Pi(M)_{top}$:* It starts with computing the activation values for the layer k (Line 1). Algorithm 2 is invoked to compute $\Pi(M)_{top}$ together with the set of conditions in the rules of $\Pi(M)_{top}$ (Line 2).
- *Computing intermediate-level rules* (Lines 4-13). This is done through a two-stage iterative process. First, it iterates over each hidden layer i (Line 4). Then, it processes each condition t in *Conditions*, it identifies the level, node index, operator, and threshold from t (Line 7). Furthermore, it evaluates the condition t given the activation value at node given the input values $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$, ...,$\mathbf{x}^{(N)}$ (Line 8) and

prepares the data for the extraction of the intermediate-rules (Line 9). Finally, it calls Algorithm 4 to compute the intermediate rules pertaining to t ($\Pi(M)_i^t$) and the set of conditions occurring in this set of rules ($Conditions_i^t$) (Line 10). Lines 11-12 do some housekeeping and it is obvious what they are doing.

- *Computing bottom-level rules,* $\Pi(M)_{bottom}$ (Lines 14-19). After completing the final iteration for computing intermediate-level rules with $i = 1$, for each condition t that occurs in the first hidden layer, the level with a value of 1, node index, operator, and threshold is extracted (Line 15). The evaluation of t given the training data set is computed (Line 16), which is then used to create the data for the bottom-rule extraction process (Line 17). The algorithm calls Algorithm 5 to compute the set of bottom-rules associated with t, $\Pi(M)_{bottom}^t$ (Line 19) which is added to $\Pi(M)$.

3.2.1 Computing Top-Level Rules

Algorithm 2 $extract_top_level(data_{top})$

1: Initializing: $\Pi(M)_{top} \leftarrow \emptyset$; $Conditions \leftarrow \emptyset$; $index \leftarrow 0$
2: $R_{top} \leftarrow C50(data_{top})$
3: **for** $r \in R_{top}$ **do**
4: $T \leftarrow getConditions(r)$; $c \leftarrow r[\text{“}ok\text{”}]/r[\text{“}cover\text{”}]$
5: $head \leftarrow potential_predict_output(r[\text{“}class\text{”}], index, c)$; $body \leftarrow \emptyset$
6: **for** $t \in T$ **do**
7: $Conditions, literal \leftarrow process_condition(t, Conditions)$
8: $body \leftarrow body \cup \{literal\}$
9: $\Pi(M)_{top} \leftarrow \Pi(M)_{top} \cup \{head \leftarrow body\}$; $index \leftarrow index + 1$
10: **return** $\Pi(M)_{top}, Conditions$

Algorithm 2 utilizes `C5.0` to compute a set of IF-THEN rules R_{top} (Line 2) given the data set $data_{top}$. Each rule r is associated with an *index* (initialized in Line 1 and incremented in Line 9) and consists of a set of arithmetic conditions T (Line 4), a predicted class label ($r[\text{“}class\text{”}]$), and a confidence value c (Line 4). This is used to constructed the class atom *head* of a top-level rule (Line 5) . The body of this rule is computed in Lines 6–8 by processing each term t in T via Algorithm 3 that returns the literal needed to be in the body and the set *Conditions* (Line 7). The meaning of Lines 9-10 are obvious.

Algorithm 3 receives an arithmetic condition t (result from `C5.0`) and a set of conditions *Condition* and processes t by identifying the operator in t as follows: (*i*) updates *Condition* if t and its opposite condition do not belong to *Conditions* (Lines 7–8); and (*ii*) creates a literal l for use in the body of top-level rules (Lines 9-11). Note that l can be of the form “*not h(i,j,*$\odot$*,value,“true”)*” (Line 11), leveraging the existing condition. This reduces the number of newly created atoms, as they can be described with the existing ones. We illustrate the execution of Algorithm 2 in the next example.

Example 1 *Given the trained model* M_{xor} *and its training set as input to Algorithm 1, Table 1 is the dataset* $data_{top}$ *computed in Line 2 of Algorithm 1.* `C5.0` *generates two rules in* $R_{top} = \{r_1, r_2\}$ *(Line 2, Algorithm 2).*

For $r = r_1$*, T contains only one condition* $t = h_2_n_0 \leq -0.37244028$*, which is added to the Conditions. We obtain the first logic rule as follows:*

$$(r_1') : potential_predict_output(0,0,1) \leftarrow h(2,0,\text{“}\leq\text{”},\text{“}-372440\text{”}, true).$$

Algorithm 3 *process_condition*$(t, Conditions)$

1: $h_{level}, n_{index}, \odot, th \leftarrow t[\text{“}level\text{”}], t[\text{“}node\text{”}], t[\text{“}operator\text{”}], t[\text{“}threshold\text{”}]$
2: **if** $\odot =$ “$\leq$” **then**
3: $\quad \ominus \leftarrow$ “$>$”
4: **else**
5: $\quad \ominus \leftarrow$ “$\leq$”
6: $opposite_t \leftarrow h_{level}, n_{index}, \ominus, th$
7: **if** $t \notin Conditions \wedge opposite_t \notin Conditions$ **then**
8: $\quad Conditions \leftarrow Conditions \cup \{t\}$
9: $l \leftarrow concatenate(h_{level}, n_{index}, \odot, th, \text{“}true\text{”})$
10: **if** $opposite_t \in Conditions$ **then**
11: $\quad l \leftarrow concatenate(\text{“}not\ \text{”}, h_{level}, n_{index}, \ominus, th, \text{“}true\text{”})$
12: **return** $Conditions, l$

For $r = r_2$, T contains $t =$ h_2_n_0 > -0.37244028. Because its opposite condition belongs to Conditions, the default negative atom of the existing condition is used, resulting a second rule as follows:

$$(r_2') : potential_predict_output(1,1,1) \leftarrow not\ h(2,0,\text{“}\leq\text{”},\text{“}-372440\text{”}, true).$$

Algorithm 2 returns $\Pi(M)_{top} = \{r_1', r_2'\}$ *and* $Conditions =$ {h_2_n_0 ≤ -0.37244028}.

3.2.2 Computing Intermediate Rules

Algorithm 4 *extract_intermediate_level*$(data_i^t, h_{level}, n_{index}, \odot, th)$

1: Initializing: $\Pi(M)_i^t \leftarrow \emptyset$; $\quad Conditions_i^t \leftarrow \emptyset$; $\quad R_i^t \leftarrow C50(data_i^t)$
2: **for** $r \in R_i^t$ **do**
3: $\quad T \leftarrow getConditions(r)$
4: $\quad class \leftarrow r[\text{“}class\text{”}]$
5: $\quad$ **if** $class = True$ **then**
6: $\quad\quad head \leftarrow concatenate(h_{level}, n_{index}, \odot, th, class)$
7: $\quad\quad body \leftarrow \emptyset$
8: $\quad\quad$ **for** $t \in T$ **do**
9: $\quad\quad\quad Conditions_i^t, literal \leftarrow process_condition(t, Conditions_i^t)$
10: $\quad\quad\quad body \leftarrow body \cup \{literal\}$
11: $\quad\quad \Pi(M)_i^t \leftarrow \Pi(M)_i^t \cup \{head \leftarrow body\}$
12: **return** $\Pi(M)_i^t, Conditions_i^t$

Algorithm 4 computes the set of intermediate rules representing the dependencies between nodes in layer i and layer $i+1$. Similar to Algorithm 2, it invokes `C5.0` to generate a set of rules R_i^t from the input set of activation values and constructs the intermediate rules from R_i^t. The key difference between this algorithm and Algorithm 2 lies in that it only processes rules in R_i^t whose prediction is the true class (Lines 5–11).

Example 2 *Continuing from Example 1, Table 2 presents the dataset $data_1^t$, where $t =$ h_2_n_0 ≤ -0.37244028.* `C5.0` *uses $data_1^t$ to generate the following three IF-THEN rules:*

(r_3) $IF\ h_1_n_1 > 0.22323501 \wedge h_1_n_1 \leq 0.74552625$ $THEN\ class = False$,
(r_4) $IF\ h_1_n_1 \leq 0.22323501$ $THEN\ class = True$,
(r_5) $IF\ h_1_n_1 > 0.74552625$ $THEN\ class = True$

Table 2: $data^t_1$ for $t = h_2_n_0 \leq -0.37244028$ (Algorithm 4)

$h_1_n_0$	$h_1_n_1$	$h_1_n_2$	$h_1_n_3$	$h_2_n_0_leq_than_minus372440$
-0.33337042	0.22323501	-0.35911334	-0.30652052	True
-0.93152505	0.74552625	0.43851086	-0.88826376	False
0.41850552	0.7016869	-0.83853734	0.4565624	False
-0.70506656	0.92262095	-0.35396752	-0.5398724	True
-0.33337042	0.22323501	-0.35911334	-0.30652052	True
-0.93152505	0.74552625	0.43851086	-0.88826376	False
0.41850552	0.7016869	-0.83853734	0.4565624	False
-0.70506656	0.92262095	-0.35396752	-0.5398724	True

Algorithm 4 processes only r_4 and r_5 and and returns $\Pi(M)^t_1 = \{r'_4, r'_5\}$ and $Conditions^t_1 = \{h_1_n_1 \leq 0.22323501, h_1_n_1 > 0.74552625\}$ where

$$(r'_4) \quad h(2,0,\text{“}\leq\text{”},\text{“}-372440\text{”},true) \leftarrow h(1,1,\text{“}\leq\text{”},223235,true).$$
$$(r'_5) \quad h(2,0,\text{“}\leq\text{”},\text{“}-372440\text{”},true) \leftarrow h(1,1,\text{“}>\text{”},745526,true).$$

3.2.3 Computing Bottom-Rules

Algorithm 5 computes the set of bottom rules $\Pi(M)^t_{bottom}$ and is similar to Algorithm 4 in that it invokes `C5.0` to compute the set R of IF-THEN rules and processes only rules that predict a True class (Line 5) to corresponding bottom-rules in $\Pi(M)^t_{bottom}$. The process of each condition of each rule in R creates rules with variables representing possible pairs of features and values (Lines 10-11) to enable the evaluation of the program using all possible inputs of a feature.

Algorithm 5 $extract_bottom_level(data^t_{input}, 1, n_{index}, \odot, th)$

1: Initializing: $\Pi(M)^t_{bottom} \leftarrow \emptyset$; $R \leftarrow C50(data^t_{input})$
2: **for** $r \in R$ **do**
3: $T \leftarrow getConditions(r)$
4: $class \leftarrow r[\text{“}class\text{”}]$
5: **if** $class = True$ **then**
6: $head \leftarrow concatenate(1, n_{index}, \odot, th, class)$
7: $body \leftarrow \emptyset$
8: **for** $t \in T$ **do**
9: $h_{level}, n_{index}, \odot, th \leftarrow t[\text{“}level\text{”}], t[\text{“}node\text{”}], t[\text{“}operator\text{”}], t[\text{“}threshold\text{”}]$
10: $atom \leftarrow input(h_{level}n_{index}, \text{“}Value\text{”}h_{level}n_{index})$
11: $arithmetic_comparison \leftarrow concatenate(\text{“}Value\text{”}h_{level}n_{index}, \odot, th)$
12: $body \leftarrow body \cup \{atom, arithmetic_comparison\}$
13: $\Pi(M)^t_{bottom} \leftarrow \Pi(M)^t_{bottom} \cup \{head \leftarrow body\}$
14: **return** $\Pi(M)^t_{bottom}$

Example 3 *The dataset $data^t_{input}$ for $t = h_1_n_1 \leq 0.22323501$ is encoded as a set of eight pairs (x_i, y_i), where $x_i = (input_feat_0, input_feat_1)$ and the label y_i is named $h_1_n_1_leq_than223235$ as follows: $\{((0,0),True),((0,1),False),((1,0),False),((1,1),False),((0,0),True),((0,1),False),((1,0),False),((1,1),False),\}$. `C5.0` generates the following rules:*

$$
\begin{array}{llll}
(r_6) & IF & input_feat_1 > 0 & THEN \quad class = False, \\
(r_7) & IF & input_feat_0 > 0 & THEN \quad class = False, \\
(r_8) & IF & input_feat_0 \leq 0 \wedge input_feat_1 \leq 0 & THEN \quad class = True
\end{array}
$$

Only rule r_8 is processed by Algorithm 5 and produces the following rules:

$$
\begin{array}{ll}
(r'_8) & h(1,1,\text{“}\leq\text{”},223235,true) \leftarrow \\
& input(input_feat_0, Value_input_feat_0), Value_input_feat_0 \leq 0, \\
& input(input_feat_1, Value_input_feat_1), Value_input_feat_1 \leq 0.
\end{array}
$$

Due to space limitation, the process for condition $t' = h_1_n_1 > 0.74552625$ is omitted. We note that the following rule is produced by Algorithm 5:

$$
\begin{array}{ll}
(r'_9) & h(1,1,\text{“}>\text{”},745526,true) \leftarrow \\
& input(input_feat_0, Value_input_feat_0), Value_input_feat_0 > 0, \\
& input(input_feat_1, Value_input_feat_1), Value_input_feat_1 > 0.
\end{array}
$$

To summarize, Algorithm 1 returns $\Pi(M_{xor}) = \{r'_1, r'_2, r'_4, r'_5, r'_8, r'_9\}$. Assuming no noise occurs, observe that

- *Class 0 is predicted if the condition at node 0 in hidden layer 2 is True, with 100% confident value (r'_1). This occurs when at least one of the conditions at node 1 in hidden layer 1 is True (r'_4 and r'_5), and this happens when the feature 0 and feature 1 of a given instance are either both ≤ 0 (r'_8) or both > 0 (r'_9).*
- *Class 1 is predicted if there is no evidence that the condition at node 0 in hidden layer 2 is True, with 100% confident value (r'_2). This happens when both conditions at node 1 in hidden layer 1 are not applicable (r'_4 and r'_5), meaning that the feature 0 and feature 1 of a given instance are not simultaneously ≤ 0 or > 0.*

Before presenting our experiments, let us remark that by construction, $\Pi(M)$ is stratified. This facilitates the proof that given an input $\mathbf{x} = \mathbf{v}$ in the form of a set of atoms $I = \{input(x_i, v_i) \mid x_i$ is a feature of $\mathbf{x}\}$, the program $\Pi(M) \cup I$ has a unique answer set. Furthermore, the size of the program grows proportionally to the size of the network.

4 Experiments

In this section, we evaluate our system on two synthetic datasets. The first dataset, named *XOR*, evaluates the performance of the output of xDNN$^{(ASP)}$ in prediction, consistency, and identifying the feature importance. It contains 1000 samples, each with 10 features, originated in the work of [29]. The 10-dimensional feature vectors are generated from a uniform distribution in $[0,1]$, and the binary class is $y_i = round(\mathbf{x}_1^{(i)}) \oplus round(\mathbf{x}_2^{(i)})$, where $\oplus$ presents the XOR operation between the 1^{st} and 2^{nd} features. The second dataset, named *Modified-XOR*, is a slight modification of the *XOR* dataset where the class label is defined by $y_i = (round(\mathbf{x}_1^{(i)}) \oplus round(\mathbf{x}_2^{(i)})) \oplus round(\mathbf{x}_3^{(i)})$. We note that for *XOR*, only x_1^i and x_2^i contribute to the value of the output, i.e., only two features are important. On the other hand, for *Modified-XOR*, three features, x_1^i, x_2^i, and x_3^i are important. This means that *Modified-XOR* is more complex than *XOR*. *Modified-XOR* was designed to investigate the influence of hidden nodes, which could contribute to the redesign of neural network architectures.

Experiments explored various neural network configurations such as the number of hidden layer nodes $\{8, 16, 32, 64, 128\}$, potential activation functions $\{tanh, relu, elu\}$, number of epochs $\{50, 100, 150, 200\}$, and batch size $\{16, 32\}$. The input layer had 10 nodes, and the output layer for binary classification had 1 node, following [28].

Recall that xDNN$^{(ASP)}$ computes $\Pi(M)$ given a trained DNN model M and its training dataset X. Given an instance $(\mathbf{x}, y)$, we say that $\Pi(M)$ correctly predicts the class y if y is a *most appropriate class* in the answer set A of $\Pi(M) \cup input(x_i, v_i)$ where y is a most appropriate class if it is the maximal element of the ordering $>$ defined by $l > l'$ if (*i*) the number of atoms of the form $potential_predict_output(l, _, _)$ is larger than the number of atoms of the form $potential_predict_output(l', _, _)$; or (*ii*) the number of atoms of the form $potential_predict_output(l, _, _)$ equals the number of atoms of the form $potential_predict_output(l', _, _)$ and l occurs in an atom with a higher confidence value. The accuracy of the program $\Pi(M)$ is defined as the percentage of the number of corrected prediction over the size of the test dataset. It reflects how well is $\Pi(M)$ comparing to M (see Tables 3 and 4). Fidelity measures the percentage between the accuracy of the extracted rules and the original model from which the rules were derived, showing how closely our extracted rules reflect the performance of the original model (see Figures 2 and 3).

We compare xDNN$^{(ASP)}$ with ECLAIRE, the best performer among the decompositional algorithms DeepRED ([30]), REM-D ([26]) and ECLAIRE ([29]). The key distinction between xDNN$^{(ASP)}$ and ECLAIRE lies in that ECLAIRE only generates a set of IF-THEN rules that map input feature conditions directly to a class. We do not include a comparison xDNN$^{(ASP)}$ SHAP ([18]) in this paper as the latter does not extract rules describing input-output dependencies. We note that in our study, our method and SHAP identify the same top features.

4.1 Accuracy, Consistency and Feature Importance

Table 3: M vs. $\Pi(M)$ in 5 configurations

Design	Hidden nodes	Epoch size	Batch	Accuracy (%)	
				M	$\Pi(M)$
1	128	200	32	96.3	94
2	128	150	32	95.2	94
3	64	200	16	96.2	93.8
4	128	150	16	95.5	93.2
5	32	200	16	95.1	92.6

To establish a baseline for our experiment, we use Weka ([16]) and create decision tree classification using 5-fold cross-validation. This baseline has accuracy of 84.4%. We then experiment with different configurations for the DNN used in this experiment. The reported designs in Table 3 are the five with accuracy over 90%. In this experiment, we use a simpler network with one hidden layer. All NNs achieved over 95% accuracy with 5-fold stratified cross-validation. xDNN$^{(ASP)}$ achieved the highest rule extraction accuracy in designs 1 and 2 at 94%, while design 5 (with 32 hidden nodes) had the lowest rule accuracy at 92.6% among them. Our results are comparable to ECLAIRE, which achieve $91.8 \pm 2.6\%$ rule accuracy using a neural network with 3 hidden layers ([29]).

Fig. 2 compares the accuracy of NNs and four rule extraction algorithms across five designs in Table 3, while Fig. 3 presents fold-level accuracy for NN, ECLAIRE and xDNN$^{(ASP)}$ on design 2. If the extracted rules closely mirror the behavior of their original NN models, their accuracies should align

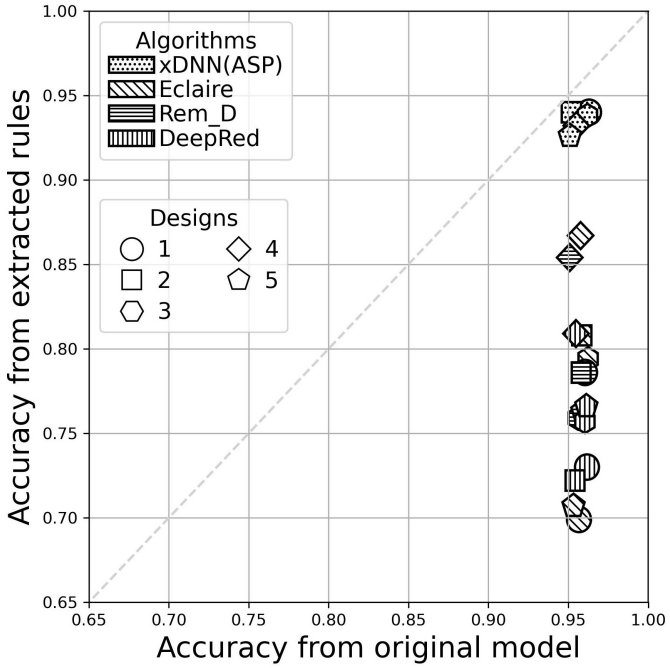


Figure 2: Comparing xDNN$^{(ASP)}$, ECLAIRE, REM-D and DeepRED.

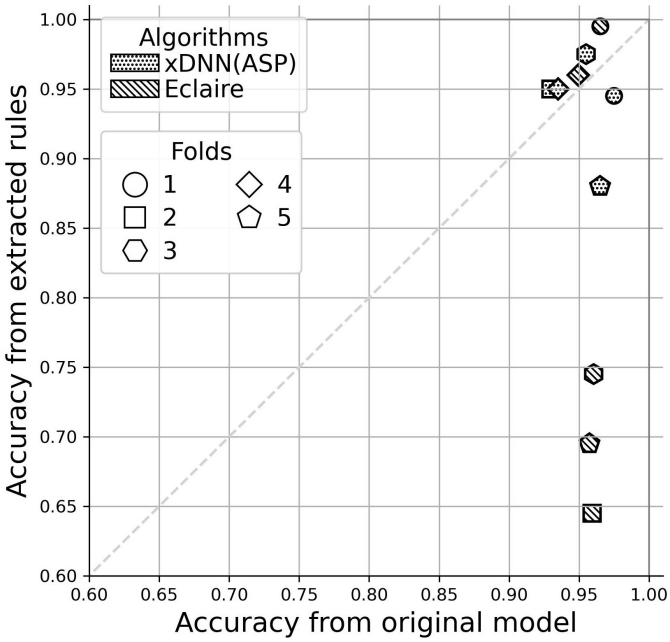


Figure 3: xDNN$^{(ASP)}$ vs. ECLAIRE wrt. 5-fold in Design 2, Table 3.

closely, meaning that each point should lie near the diagonal line. In Fig. 2, while all NN models performed well across the five designs, the rule accuracy of xDNN$^{(ASP)}$ consistently outperformed the others, demonstrating high fidelity. Fig. 3 shows that, while ECLAIRE reached 99.5% accuracy in one fold, its performance dropped to about 65% in another, indicating inconsistency within the same design. Thus, ECLAIRE may need more exploration of NN designs to improve both rule extraction and model

accuracy. In contrast, xDNN$^{(\texttt{ASP})}$ delivered more stable results across all folds, despite slightly lower peak accuracy. We define the importance of a feature x_i as the percentage between the number of occurrences of $input(x_i,_)$ in $\Pi(M)$ compared to the total number of occurrences of all features in $\Pi(M)$. Fig. 4 shows these percentages for each input feature in the extracted rules across five folds for design 1 (highest accuracy) and design 5 (lowest accuracy). The 1^{st} and 2^{nd} features contribute most in both designs, reflecting their role in label generation.

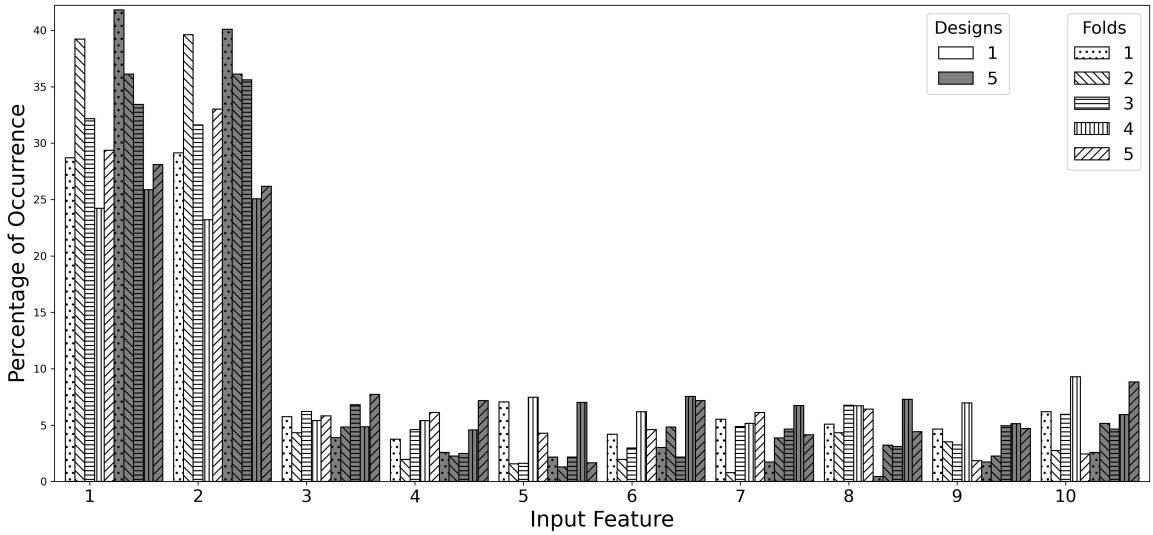


Figure 4: Feature importance in $\Pi(M)$ across 5 folds in the designs 1 and 2 from Table 3

4.2 Hidden Node Impact and Neural Network Redesign

Table 4: M vs. $\Pi(M)$ in five designs

Design	Hidden nodes	Epoch size	Batch	Accuracy (%)	
				M	$\Pi(M)$
1	128	200	16	91.9	83.4
2	16	200	16	90.8	82.6
3	64	200	16	92.3	82.4
4	32	200	32	90.3	80.7
5	**10**	200	16	**92.1**	**82.8**

Similar to the first experiment, we create a baseline for our experiment using Weka. Since the dataset is more complex, the baseline has the accuracy of only 52.3%. We report the top four designs for DNN that achieves the accuracy better than 90% (Lines 1-4, Table 4). The rule extraction accuracy of xDNN$^{(\texttt{ASP})}$ ranged from 80.7% (32 nodes) to 83.4% (128 nodes). A special design (#5) uses only 10 nodes but boasts the 2^{nd} highest accuracy in both trained network (M) and logic program $\Pi(M)$. This choice is due to the discovery of hidden node impact, discussed next.

Similar to feature importance, we define the impact score of a hidden node j at layer i, denoted by $impact(h_i^j)$, as the product of two numbers based on its role: (a) the number of occurrences of atoms $h/5$ with i and j in the head which indicates how much it is influenced by others, and (b) the number of occurrences of such atoms in the body which indicates its influences on other atoms. Figure 5 ranks the top 20 hidden nodes by impact score percentage, calculated by $(impact(h_i^j)/\sum_{j=1}^{n} impact(h_i^j) * 100)$,

where $n = 128$ for Fold 1 of Design 1 (Table 4). It shows that only a small subset of nodes in the hidden layers significantly contributes to the output of the network. This trend is also observed in all other configurations. This stipulates us to reduce the number of hidden nodes in Design 1 from 128 to 10 for an experiment. As we can see, this does not result in a reduction of accuracy of the network or the output of xDNN$^{(ASP)}$ (Li. 5, Tab. 4).

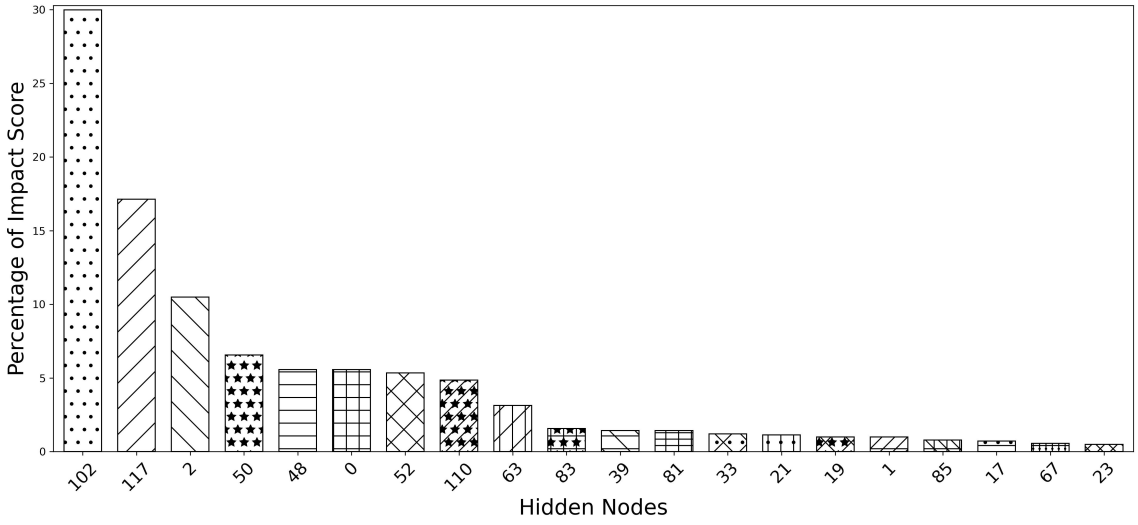


Figure 5: Top 20 hidden nodes' impact in fold 1, design 1 (Table 4)

5 Conclusion

We propose a system for understanding DNN, xDNN$^{(ASP)}$, that extracts answer set programs from trained models. The extracted program maintains a high-level of accuracy of the original model and allows for the identification of features that are important to the prediction of the network that is consistent with statistical method. In addition, the program provides means for the evaluation of impacts of hidden nodes on the output of the network, which can be used to simplify the design of the original network. Our experiments show that xDNN$^{(ASP)}$ performs better than previously developed decompositional systems for understanding DNN such as ECLAIRE, DeepRED, and REM-D in several aspects. We note that the size of the program generated by xDNN$^{(ASP)}$ is linear in the size of the network. It might therefore be challenging if one wants to visually inspect the program. However, this is reasonable for analyzing purpose such as identifying important features or studying impacts of a hidden node or the interactions between the nodes or using the program to generate explanations for the outputs as in [2]. Future research could focus on reducing computational costs. One potential approach is to examine the impact of groups of hidden nodes rather than individual ones. Another interesting direction is to explore specialized neural network architectures, such as Convolutional Neural Networks, to tackle classification problems involving non-numerical data, such as images.

Acknowledgments

Ly Trieu is supported by Los Alamos National Laboratory under the award number 047229. Son Tran was also partially supported by NSF grants 1914635 and 2151254.

References

[1] Sajid Ali, Tamer Abuhmed, Shaker El-Sappagh, Khan Muhammad, Jose M Alonso-Moral, Roberto Confalonieri, Riccardo Guidotti, Javier Del Ser, Natalia Díaz-Rodríguez & Francisco Herrera (2023): *Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence*. *Information Fusion* 99, p. 101805, doi:10.1016/j.inffus.2023.101805.

[2] Mario Alviano, Ly Ly Trieu, Tran Cao Son & Marcello Balduccini (2024): *The XAI system for answer set programming xASP2*. *Journal of Logic and Computation* 34(8), pp. 1500–1525, doi:10.1093/logcom/exae036.

[3] Robert Andrews, Joachim Diederich & Alan B Tickle (1995): *Survey and critique of techniques for extracting rules from trained artificial neural networks*. *Knowledge-based systems* 8(6), pp. 373–389, doi:10.1016/0950-7051(96)81920-4.

[4] Plamen P Angelov, Eduardo A Soares, Richard Jiang, Nicholas I Arnold & Peter M Atkinson (2021): *Explainable artificial intelligence: an analytical review*. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 11(5), p. e1424, doi:10.1002/widm.1424.

[5] Jason Bell (2020): *Machine learning: hands-on for developers and technical professionals*. John Wiley & Sons, doi:10.1002/9781119642183.

[6] Parteek Bhatia (2019): *Data mining and data warehousing: principles and practical techniques*. Cambridge University Press, doi:10.1017/9781108635592.

[7] Przemyslaw Biecek & Tomasz Burzykowski (2021): *Local interpretable model-agnostic explanations (LIME)*. *Explanatory Model Analysis Explore, Explain and Examine Predictive Models* 1, pp. 107–124, doi:10.1201/9780429027192.

[8] Christopher M Bishop & Nasser M Nasrabadi (2006): *Pattern recognition and machine learning*. 4, Springer.

[9] Hung-Ming Chiu, Shih-En Lin, Yen-Wei Chu & Chih-Jung Chen (2025): *Waikato Environment for Knowledge Analysis (WEKA) as a Data Analysis Method Identifying Potential Hematological Parameters for Early Diagnosis of Cervical Cancer*. *In Vivo* 39(2), pp. 1042–1053, doi:10.21873/invivo.13909.

[10] Jorge Fandinno & Claudia Schulz (2019): *Answering the "why" in answer set programming–A survey of explanation approaches*. *Theory and Practice of Logic Programming* 19(2), pp. 114–203, doi:10.1017/S1471068418000534.

[11] Joao Ferreira, Manuel de Sousa Ribeiro, Ricardo Goncalves & Joao Leite (2022): *Looking inside the black-box: Logic-based explanations for neural networks*. In: *Proceedings of the international conference on principles of knowledge representation and reasoning*, 19, pp. 432–442, doi:10.24963/kr.2022/45.

[12] M. Gelfond & V. Lifschitz (1990): *Logic programs with classical negation*. In D. Warren & Peter Szeredi, editors: *7th ICLP*, pp. 579–597.

[13] Ian Goodfellow, Yoshua Bengio & Aaron Courville (2016): *Deep Learning*. The MIT Press.

[14] Riccardo Guidotti (2024): *Counterfactual explanations and how to find them: literature review and benchmarking*. *Data Mining and Knowledge Discovery* 38(5), pp. 2770–2824, doi:10.1007/s10618-022-00831-6.

[15] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti & Dino Pedreschi (2018): *A survey of methods for explaining black box models*. *ACM computing surveys (CSUR)* 51(5), pp. 1–42, doi:10.1145/3236009.

[16] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann & Ian H Witten (2009): *The WEKA data mining software: an update*. *ACM SIGKDD explorations newsletter* 11(1), pp. 10–18, doi:10.1145/1656274.1656278.

[17] Max Kuhn & Kjell Johnson (2013): *Applied predictive modeling*. 26, Springer, doi:10.1007/978-1-4614-6849-3.

[18] Scott M Lundberg & Su-In Lee (2017): *A unified approach to interpreting model predictions*. *Advances in neural information processing systems* 30.

[19] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot & Edouard Duchesnay (2011): *Scikit-learn: Machine learning in Python*. the Journal of machine Learning research 12, pp. 2825–2830.

[20] J. Ross Quinlan (1986): *Induction of decision trees*. Machine learning 1, pp. 81–106, doi:10.1007/BF00116251.

[21] J Ross Quinlan (1993): *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., doi:10.1007/BF00993309.

[22] J Ross Quinlan (2004): *Data mining tools See5 and C5.0*. www.rulequest.com/see5-info.html.

[23] M. Mazhar Rathore, Syed Attique Shah, Dhirendra Shukla, Elmahdi Bentafat & Spiridon Bakiras (2021): *The Role of AI, Machine Learning, and Big Data in Digital Twinning: A Systematic Literature Review, Challenges, and Opportunities*. IEEE Access 9, pp. 32030–32052, doi:10.1109/ACCESS.2021.3060863.

[24] Richard J Roiger (2017): *Data mining: a tutorial-based primer*. Chapman and Hall/CRC, doi:10.1201/9781315382586.

[25] Wojciech Samek & Klaus-Robert Müller (2019): *Towards explainable artificial intelligence*. Explainable AI: interpreting, explaining and visualizing deep learning, pp. 5–22, doi:10.1007/978-3-030-28954-6_1.

[26] Zohreh Shams, Botty Dimanov, Sumaiyah Kola, Nikola Simidjievski, Helena Andres Terre, Paul Scherer, Urška Matjašec, Jean Abraham, Pietro Liò & Mateja Jamnik (2021): *REM: An integrative rule extraction methodology for explainable data analysis in healthcare*. medRxiv, pp. 2021–01, doi:10.1101/2021.01.25.21250459.

[27] Sanjay Sharma, C Rama Krishna & Sanjay K Sahay (2019): *Detection of advanced malware by machine learning techniques*. In: Soft Computing: Theories and Applications: Proceedings of SoCTA 2017, Springer, pp. 333–342, doi:10.1007/978-981-13-0589-4_31.

[28] Pang-Ning Tan, Michael Steinbach & Vipin Kumar (2016): *Introduction to data mining*. Pearson Education India.

[29] Mateo Espinosa Zarlenga, Zohreh Shams & Mateja Jamnik (2021): *Efficient decompositional rule extraction for deep neural networks*. arXiv preprint arXiv:2111.12628.

[30] Jan Ruben Zilke, Eneldo Loza Mencía & Frederik Janssen (2016): *Deepred–rule extraction from deep neural networks*. In: Discovery Science: 19th International Conference, DS 2016, Bari, Italy, October 19–21, 2016, Proceedings 19, Springer, pp. 457–473, doi:10.1007/978-3-319-46307-0_29.

Implementing the First-Order Logic of Here and There

Jens Otten
University of Pernambuco
Recife, Brazil
jeotten@leancop.de

Torsten Schaub
University of Potsdam
Potsdam, Germany
torsten@cs.uni-potsdam.de

We present automated theorem provers for the first-order logic of here and there (HT). They are based on a native sequent calculus for the logic of HT and an axiomatic embedding of the logic of HT into intuitionistic logic. The analytic proof search in the sequent calculus is optimized by using free variables and skolemization. The embedding is used in combination with sequent, tableau and connection calculi for intuitionistic first-order logic. All provers are evaluated on a large benchmark set of first-order formulas, providing a foundation for the development of more efficient HT provers.

1 Introduction

The *Logic of Here and There* (HT; [13, 12]) is an intermediate logic that is between classical logic (CL) and intuitionistic logic (IL). HT is weaker than classical logic as — just like intuitionistic logic — it does not validate the law of excluded middle $A \vee \neg A$, giving it a more constructive character. However, it is stronger than intuitionistic logic as the so-called *weak law of excluded middle* $\neg A \vee \neg\neg A$ is valid in HT.

HT plays a central role and provides the fundamental logical foundations for *Answer Set Programming* (ASP; [15]). Specifically, the stable models of a logic program [10] can be characterized by a simple, minimal model construction within HT, and are termed equilibrium models [27]. This characterization offers several advantages. Notably, logical equivalence in HT implies substitutability without altering the equilibrium models of the containing formula [16]. This property is crucial as it permits the simplification of a logic program segment independently of its context. Furthermore, HT facilitates the characterization of ASP extensions incorporating foreign language constructs and inferences [1, 6, 2, 5]. A first tableau system for HT and its extension for computing equilibrium models was presented in [28].

While these works provide essential cornerstones for the advancement of ASP, they are propositional, contrasting with the first-order nature of ASP encodings. To address this, first-order extensions of HT were introduced in [29], and some of the above works lifted to the first-order case [17]. However, direct automated reasoning within HT remains undeveloped. Current approaches, such as *anthem* [8, 7], which determines HT equivalence of two ASP encodings, rely on translations into Boolean first-order logic and employ conventional theorem provers for classical first-order logic.

We address this shortcoming by developing first-order theorem proving technology directly within HT, leading to four new theorem provers for HT. The first prover, leanHaT, is based on a first-order sequent calculus for HT. The other three provers, ileanSeP-HT, ileanTAP-HT, nanoCoP-i-HT, are based on an axiomatic embedding of first-order HT into first-order intuitionistic logic and use the intuitionistic theorem provers ileanSeP, ileanTAP and nanoCoP-i, respectively. These intuitionistic provers are based on a single-succedent intuitionistic sequent calculus, a prefixed tableau calculus and a prefixed non-clausal connection calculus, respectively.

All four developed theorem provers for HT follow the *lean methodology* for automated theorem provers as first propagated by the theorem provers lean*TAP* [3] and leanCoP [25] for classical first-order logic; they are implemented by using very compact yet efficient Prolog code.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 453–468, doi:10.4204/EPTCS.439.31

© J. Otten & T. Schaub
This work is licensed under the
Creative Commons Attribution License.

The paper is structured as follows. Section 2 presents the sequent calculus LHT for HT and the axiomatic embedding of HT into intuitionistic logic. Section 3 describes proof search optimizations and methods. It includes techniques to prune the search space in the sequent calculus LHT, such as free variables and skolemization, and an optimized axiomatic embedding of HT into intuitionistic logic, which drastically reduces the number of axioms. We also provide details of the prefixed non-clausal connection calculus for intuitionistic logic, which is used by nanoCoP-i-HT and among the most efficient proof search calculi for intuitionistic logic. Section 4 presents implementation details of the four developed theorem provers for HT. A comprehensive experimental evaluation of the theorem provers for HT on a large set of benchmark formulas is provided in Section 5, before the paper concludes with a summary and outlook in Section 6.

2 Preliminaries

The following section describes syntax, semantics and a sequent calculus for the logic of HT, as well as an axiomatic embedding of the logic of HT into intuitionistic logic.

2.1 Syntax and Semantics of HT

The logic of HT has the same *syntax* as classical and intuitionistic logic, but a different semantics. Propositional formulas are defined inductively from atomic (propositional) formulas (p,q) and the connectives $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication) and $\neg$ (negation). This definition is extended to first-order logic by adding first-order predicates, variables, terms and the quantifiers $\forall$ (universal) and $\exists$ (existential). A *term* is either a variable (x,y,z) or $f(t_1,\ldots,t_n)$, in which f is an n-ary function symbol and $t_1,\ldots,t_n$ are terms (for $n \geq 0$). A *(first-order) formula* (A,B,F,G,H) is defined inductively and is either an atomic formula $P(t_1,\ldots,t_n)$ for an n-ary predicate symbol P and terms $t_1,\ldots,t_n$ (for $n \geq 0$), or a formula of the form $G \wedge H$, $G \vee H$, $G \rightarrow H$, $\neg G$, $\forall x\, G$ and $\exists x\, G$ for formulas G, H and variable x. A literal L has the form A or $\neg A$ for an atomic formula A. The *free variables* in a formula are those not in the scope of a quantifier. $A[x\backslash t]$ denotes the formula obtained by substituting the term t for all free variable occurrences of x in A. $\forall x_1,\ldots,x_n$ and $\exists x_1,\ldots,x_n$ are short for $\forall x_1 \ldots \forall x_n$ and $\exists x_1 \ldots \exists x_n$, respectively.

The *semantics* of HT is defined by an intuitionistic Kripke-structure with two worlds h, t ("here" and "there") with constant individual domain and a reflexive and transitive accessibility relation between these two worlds. The connectives and quantifiers are interpreted classically in each single world except for implication and negation, which are interpreted with respect to both worlds. A formula that is true in h is also true in t. See [17] for more details.

2.2 Sequent Calculus for HT

A *sequent calculus* allows us to derive all *valid* formulas of a given logic. Sequent calculi for classical and intuitionistic first-order logic were first introduced by Gentzen [11]. A formal (sequent) calculus consists of rules and axioms. Each rule has one or more premises and one conclusion, axioms have one conclusion but no premises. For examples, the sequent calculus for classical logic consists of two rules for each connective and quantifier and one axiom. The premises and conclusions are *sequents* of the form $\Gamma \vdash \Delta$ where Γ and Δ are multisets of (first-order) formulas. Γ, A or A, Γ is short for $\Gamma \cup \{A\}$. A *sequent proof* of a (first-order) formula F is a derivation tree of the sequent $\vdash F$ (i.e. of $\{\} \vdash \{F\}$) applying the rules of the sequent calculus, in which all leaves are axioms.

A sequent calculus for the first-order logic of HT was given by Mints [18]. It consist of two axioms and two rules for each of the connectives and quantifiers $\wedge$, $\vee$, $\rightarrow$, $\forall$ and $\exists$. Additionally, there are altogether 12 rules for negation $\neg$ in combination with each of the connectives and quantifiers.

Definition 1 (Sequent Calculus LHT for HT) *The axioms and rules of the sequent calculus LHT for HT are given in Figure 1 [18].*[1] *t is an arbitrary term and for the rules marked with * the* Eigenvariable condition *has to be respected: the Eigenvariable z must not occur in the conclusion, i.e. in Γ, Δ, or A.*

Then, a formula F is valid in HT if, and only if, there is a sequent proof for F in LHT [18].

$$\frac{}{\Gamma, A \vdash A, \Delta}\ \text{axiom}_1 \qquad \frac{}{\Gamma, A, \neg A \vdash \Delta}\ \text{axiom}_2$$

$$\frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta}\ \wedge\text{-left}\ \textit{(r1)} \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma \vdash B, \Delta}{\Gamma \vdash A \wedge B, \Delta}\ \wedge\text{-right}\ \textit{(r8)}$$

$$\frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta}\ \vee\text{-left}\ \textit{(r9)} \qquad \frac{\Gamma \vdash A, B, \Delta}{\Gamma \vdash A \vee B, \Delta}\ \vee\text{-right}\ \textit{(r2)}$$

$$\frac{\Gamma, \neg A \vdash \Delta \qquad \Gamma \vdash A, \neg B, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \rightarrow B \vdash \Delta}\ \rightarrow\text{-left}\ \textit{(r14)}$$

$$\frac{\Gamma, A \vdash B, \Delta \qquad \Gamma, \neg B \vdash \neg A, \Delta}{\Gamma \vdash A \rightarrow B, \Delta}\ \rightarrow\text{-right}\ \textit{(r13)}$$

$$\frac{\Gamma, \neg A \vdash \Delta \qquad \Gamma, \neg B \vdash \Delta}{\Gamma, \neg(A \wedge B) \vdash \Delta}\ \neg\wedge\text{-left}\ \textit{(r10)} \qquad \frac{\Gamma \vdash \neg A, \neg B, \Delta}{\Gamma \vdash \neg(A \wedge B), \Delta}\ \neg\wedge\text{-right}\ \textit{(r3)}$$

$$\frac{\Gamma, \neg A, \neg B \vdash \Delta}{\Gamma, \neg(A \vee B) \vdash \Delta}\ \neg\vee\text{-left}\ \textit{(r4)} \qquad \frac{\Gamma \vdash \neg A, \Delta \qquad \Gamma \vdash \neg B, \Delta}{\Gamma \vdash \neg(A \vee B), \Delta}\ \neg\vee\text{-right}\ \textit{(r11)}$$

$$\frac{\Gamma, \neg B \vdash \neg A, \Delta}{\Gamma, \neg(A \rightarrow B) \vdash \Delta}\ \neg\rightarrow\text{-left}\ \textit{(r5)} \qquad \frac{\Gamma, \neg A \vdash \Delta \qquad \Gamma \vdash \neg B, \Delta}{\Gamma \vdash \neg(A \rightarrow B), \Delta}\ \neg\rightarrow\text{-right}\ \textit{(r12)}$$

$$\frac{\Gamma \vdash \neg A, \Delta}{\Gamma, \neg\neg A \vdash \Delta}\ \neg\neg\text{-left}\ \textit{(r6)} \qquad \frac{\Gamma, \neg A \vdash \Delta}{\Gamma \vdash \neg\neg A, \Delta}\ \neg\neg\text{-right}\ \textit{(r7)}$$

$$\frac{\Gamma, A[x\backslash t], \forall x A \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\ \forall\text{-left}\ \textit{(r25)} \qquad \frac{\Gamma \vdash A[x\backslash z], \Delta}{\Gamma \vdash \forall x A, \Delta}\ \forall\text{-right}^*\ \textit{(r21)}$$

$$\frac{\Gamma, A[x\backslash z] \vdash \Delta}{\Gamma, \exists x A \vdash \Delta}\ \exists\text{-left}^*\ \textit{(r22)} \qquad \frac{\Gamma \vdash \exists x A, A[x\backslash t], \Delta}{\Gamma \vdash \exists x A, \Delta}\ \exists\text{-right}\ \textit{(r26)}$$

$$\frac{\Gamma, \neg A[x\backslash z] \vdash \Delta}{\Gamma, \neg(\forall x A) \vdash \Delta}\ \neg\forall\text{-left}^*\ \textit{(r19)} \qquad \frac{\Gamma \vdash \neg(\forall x A), \neg A[x\backslash t], \Delta}{\Gamma \vdash \neg(\forall x A), \Delta}\ \neg\forall\text{-right}\ \textit{(r23)}$$

$$\frac{\Gamma, \neg A[x\backslash t], \neg(\exists x A) \vdash \Delta}{\Gamma, \neg(\exists x A) \vdash \Delta}\ \neg\exists\text{-left}\ \textit{(r24)} \qquad \frac{\Gamma \vdash \neg A[x\backslash z], \Delta}{\Gamma \vdash \neg(\exists x A), \Delta}\ \neg\exists\text{-right}^*\ \textit{(r20)}$$

Figure 1: The sequent calculus LHT for the first-order logic of HT.

[1] The $\neg\vee$-right-rule of the original calculus was slightly corrected by moving the formulas $\neg A$ and $\neg B$ from the left to the right side in the sequent of the premises. Otherwise, for example, the formula $(p \vee p) \rightarrow p$, which is valid in HT, is not provable and the formula $\neg p \vee \neg(p \vee p)$, which is not valid in HT, is provable.

Example 1 *Consider the formula $F_1 = (p \to q) \vee (q \to p)$, which is problem 16 of Pelletier's "75 problems for theorem proving" [30]. A sequent proof for F_1 in LHT is given below, thus F_1 is valid in HT.*

$$\dfrac{\dfrac{\dfrac{}{p,q \vdash p,q}\,axiom_1 \quad \dfrac{}{p,\neg p \vdash \neg q,q}\,axiom_2}{p \vdash q,\ q \to p}\,{\to}\text{-}right \quad \dfrac{\dfrac{}{\neg q,q \vdash p,\neg p}\,axiom_2 \quad \dfrac{}{\neg q,\neg p \vdash \neg q,\neg p}\,axiom_1}{\neg q \vdash \neg p,\ q \to p}\,{\to}\text{-}right}{\dfrac{\vdash p \to q,\ q \to p}{\vdash (p \to q) \vee (q \to p)}\,{\vee}\text{-}right}\,{\to}\text{-}right$$

Consider the formula $F_2 = \exists y \forall x\,(p(y) \to p(x))$, which is problem 18 of Pelletier [30]. A sequent proof for F_2 in LHT is given below. ax_1 and ax_2 are $axiom_1$ and $axiom_2$, respectively. For readability, the formula F_2 has been omitted from the right side of all four leaf sequents and the formula $p(a) \to p(b)$ has been omitted from the sequents of the main right branch of the proof. Thus, F_2 is valid in HT.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p(a),p(b) \vdash p(b),p(c)}\,ax_1 \quad \dfrac{}{\neg p(b),p(b) \vdash \neg p(a),p(c)}\,ax_2}{p(b) \vdash F_2,\ p(a) \to p(b),\ p(c)}\,{\to}\text{-}right \quad \dfrac{\dfrac{\dfrac{\dfrac{}{p(c),\neg p(c) \vdash p(d),\neg p(b)}\,ax_2 \quad \dfrac{}{\neg p(d),\neg p(c) \vdash \neg p(c),\neg p(b)}\,ax_1}{\neg p(c) \vdash F_2,\ p(c) \to p(d),\ \neg p(b)}\,{\to}\text{-}right}{\neg p(c) \vdash F_2,\ \forall x\,(p(c) \to p(x)),\ \neg p(b)}\,{\forall}\text{-}right^*}{\neg p(c) \vdash \exists y \forall x\,(p(y) \to p(x)),\ \neg p(b)}\,{\exists}\text{-}right}{\vdash \exists y \forall x\,(p(y) \to p(x)),\ p(b) \to p(c),\ p(a) \to p(b)}\,{\to}\text{-}right$$

$$\dfrac{\dfrac{\dfrac{\vdash \exists y \forall x\,(p(y) \to p(x)),\ \forall x\,(p(b) \to p(x)),\ p(a) \to p(b)}{\vdash \exists y \forall x\,(p(y) \to p(x)),\ p(a) \to p(b)}\,{\exists}\text{-}right}{\vdash \exists y \forall x\,(p(y) \to p(x)),\ \forall x\,(p(a) \to p(x))}\,{\forall}\text{-}right^*}{\vdash \exists y \forall x\,(p(y) \to p(x))}\,{\exists}\text{-}right$$

Neither of the two formulas is valid in intuitionistic logic, but they are both valid in classical logic.

2.3 Axiomatic Embedding of HT

The logic of HT can be axiomatized by embedding it into intuitionistic logic. This axiomatization is specified by a set of axioms $A_1, A_2, \ldots$ that are added to a given formula F, such that F is valid in HT iff $(A_1 \wedge A_2 \wedge \ldots) \to F$ is valid in intuitionistic logic.

Definition 2 (Axiomatic Embedding of HT) *The following two axiom schemas specify the set of axioms of the* axiomatic embedding of HT *into intuitionistic logic [17, 18].*

$$G \vee (G \to H) \vee \neg H \qquad \textit{(HOS)}$$
$$\exists \vec{x}\,(G \to \forall \vec{x}\,G) \qquad \textit{(SQHT)}$$

G and H are arbitrary (first-order) formulas, which might contain free variables and $\vec{x}$ is a set $x_1, \ldots, x_n$ of arbitrary variables in G. Furthermore, universal closure is applied to the axiom formulas of these schemas. More precisely, if F_A is a formula according to one of the axiom schemas and $z_1, \ldots, z_m$ are the free variables in F_A, then $\forall z_1, \ldots, z_m F_A$ is the resulting axiom.

Then, a formula F is valid in HT if, and only if, the formula F with added *HOS* and *SQHT* axioms is valid in intuitionistic logic.

Example 2 *For $F_1 = (p \to q) \vee (q \to p)$, the formulas $(p \vee (p \to q) \vee \neg q)$ and $(q \vee (q \to p) \vee \neg p)$ are instantiations of the axiom schema HOS. For $F_2 = \exists y \forall x\,(p(y) \to p(x))$ the formulas $\exists x\,(p(x) \to \forall y\,p(y))$ and $\exists x\,(\neg p(x) \to \forall y \neg p(y))$ are instantiations of the axiom schema SQHT.*

3 Proof Search

The presented sequent calculus LHT and the axiomatic embedding of HT in combination with a proof search calculus for intuitionistic logic allow us to perform *proof search*, i.e. finding a proof for the validity of a formula in the logic of HT. For an *efficient* proof search a few additional optimizations are necessary, which are described in this section.

For the axiomatic embedding of HT, we need to determine the validity of a formula in *intuitionistic logic*. One of the most efficient proof methods for intuitionistic logic (see also the results in Section 5) is the intuitionistic non-clausal connection calculus, for which we therefore also give a short introduction.

3.1 Proof Search in the Sequent Calculus LHT

The following optimizations lead to a more efficient proof search (already) for the *propositional* fragment of the sequent calculus LHT introduced in Section 2.2.

- **Bottom-up proof search.** The sequent calculus can be used in a top-down way to derive all valid formulas. In order to find a sequent proof for a formula F, it is more efficient to start with the sequent $\vdash F$ and apply the rules of LHT in a *bottom-up* (or *analytic*) way until all leaves of the derivation tree end in an axiom of LHT. In the following, we assume such an analytic proof search.
- **No backtracking for invertible rules.** If more than one rule can be applied to a sequent during the proof search, all applicable (alternative) rules have to be applied on backtracking (i.e. when a previously applied rule does not lead to a proof). For example, for the sequent $p \wedge q \vdash p \wedge q$ the rules $\wedge$-left as well as $\wedge$-right are applicable. These choices result in a high amount of backtracking.

  This backtracking is not necessary for invertible rules. A rule is called *invertible* iff whenever the conclusion of a rule is provable so are all its premises (the opposite direction obviously holds).

  For example, the following (propositional) rules of the intuitionistic sequent calculus LJ [11] are not invertible and, hence, require backtracking (consider, e.g., a proof for the sequent $q \vdash p \vee q$):

  $$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\text{-right} \qquad\qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\text{-right}$$

  All propositional rules of the sequent calculus LHT are invertible (as well as the quantifier rules with Eigenvariable condition marked with *) [18]. Therefore, no backtracking over alternative rule applications is necessary for these (invertible) rules.
- **Apply non-splitting rules first.** If a sequent rule has more than one premise, the proof search splits into several branches, all of which must be proved. Applying the non-splitting rules first leads to a slightly more efficient proof search as proofs and failed proof attempts are found faster.
- **Restricting axiom formulas.** As in the classical and intuitionistic sequent calculi, the formula A in the axioms can be restricted to literals and atomic formulas. It can be shown that A can be restricted to literals in axiom_1 and to atomic formulas in axiom_2 (allowing literals for A in axiom_2 is redundant as $\neg\neg$-left and axiom_1 could be applied in this case). This leads to two new axioms:

  $$\frac{}{\Gamma, G \vdash G, \Delta}\ \text{axiom}_1 \qquad\qquad \frac{}{\Gamma, H, \neg H \vdash \Delta}\ \text{axiom}_2$$

  where G is a literal and H is an atomic formula. Restricting axiom formulas in this way is necessary when using free variables and unification whenever branches are closed by axioms (see below).

These proof search optimizations lead to a *decision procedure* for the *propositional* fragment of HT. This follows from the fact that in each (analytic) proof step the size of at least one formula in a sequent is reduced to one or more smaller formulas, whereas the size and number of all other formulas in the sequent remain unchanged. The *size of a formula F* is its number of atomic formulas and connectives.

The first-order logic of HT — like first-order intuitionistic and first-order classical logic — is *not* decidable, due to the quantifier rules $\forall$-left, $\exists$-right, $\neg\forall$-right and $\neg\exists$-left. Nevertheless, the following optimizations lead to a more efficient proof search for first-order formulas in the sequent calculus LHT.

- **Free variables.** In order to avoid "guessing" an appropriate term t when applying one of the rules $\forall$-left, $\exists$-right, $\neg\forall$-right and $\neg\exists$-left (which would result in a huge search space), we use free variables instead. In this way we delay the choice for t until axioms are reached, where now term unification is used to make G and H (see above) identical resulting in an appropriate term for t. We replace the rules $\forall$-left and $\exists$-right by the following new *free variable (quantifier) rules*:

$$\frac{\Gamma, A[x\backslash x^*], \forall x A \vdash \Delta}{\Gamma, \forall x A \vdash \Delta}\ \forall\text{-left} \qquad \frac{\Gamma \vdash \exists x A, A[x\backslash x^*], \Delta}{\Gamma \vdash \exists x A, \Delta}\ \exists\text{-right}$$

  where x^* is a new/fresh variable. The rules $\neg\forall$-right and $\neg\exists$-left are replaced accordingly.

- **Skolemization of Eigenvariables.** In order to check the Eigenvariable condition when using free variables and term unification, a *dynamic skolemization* is used that detects whenever the order of quantifier rule applications violates the Eigenvariable condition (by the occurs check of the term unification). We replace the rules $\forall$-right* and $\exists$-left* by the following new rules:

$$\frac{\Gamma \vdash A[x\backslash t^*], \Delta}{\Gamma \vdash \forall x A, \Delta}\ \forall\text{-right}^* \qquad \frac{\Gamma, A[x\backslash t^*] \vdash \Delta}{\Gamma, \exists x A \vdash \Delta}\ \exists\text{-left}^*$$

  where t^* is the Skolem term $f^*(x_1, \ldots, x_n)$, in which f^* is a new function symbol and $x_1, \ldots, x_n$ are all free variables in Γ and Δ. The rules $\neg\forall$-left* and $\neg\exists$-right* are replaced accordingly.

- **Iterative deepening.** The application of the *free variable quantifier* rules ($\forall$-left, $\exists$-right, $\neg\forall$-right and $\neg\exists$-left) might lead to non-terminating proof branches. In order to regain completeness, an iterative deepening over the number of free variables that are allowed on a branch in a proof derivation is conducted. If there are no free variable quantifiers in the given formula, no iterative deepening is conducted leading to a decision procedure for such first-order formulas.

3.2 A Restricted Axiomatic Embedding of HT

In the axiom schemas for HT introduced in Section 2.3, G and H are arbitrary formulas. This leads to an infinite number of axioms that have to be added (step by step) to the given formula to be proved, resulting in a huge search space. In order to keep the number of these axioms low, we restrict G to literals and H to atomic formulas. Furthermore, we can restrict the axiom schema *HOS* to formulas $G \neq H$.

Definition 3 (Restricted Axiomatic Embedding of HT) *Let F be a (first-order) formula, $\mathscr{P}$ the set of predicate symbols in F and $\langle P \rangle$ the arity of the predicate symbol $P \in \mathscr{P}$. The following set of axioms for a formula F specifies a* restricted axiomatic embedding of HT *into intuitionistic logic.*

$$\begin{aligned} \mathit{AxiomsHT} := \{ &\forall x_1, \ldots, x_n \forall y_1, \ldots, y_m\, (G \vee (G \rightarrow H) \vee \neg H),\ \exists x_1, \ldots, x_n\, (G \rightarrow \forall x_1, \ldots, x_n\, G) \ \mid \\ &G = P(x_1, ..., x_n) \text{ or } G = \neg P(x_1, ..., x_n),\ H = P'(y_1, ..., y_m) \text{ for } P, P' \in \mathscr{P} \text{ with } P \neq P',\ n = \langle P \rangle,\ m = \langle P' \rangle \} \end{aligned}$$

The evaluation in Section 5 reveals whether this is a reasonable restriction. If both, G and H are restricted to atomic formulas, the weak law of excluded middle $\neg p \vee \neg\neg p$ is not provable any more.

Example 3 *Consider the formula $F_1 = (p \to q) \vee (q \to p)$. The set AxiomsHT includes six axioms (for $G \in \{p, q, \neg p, \neg q\}$ and $H \in \{p, q\}$ with $G \neq H$), of which either $(p \vee (p \to q) \vee \neg q)$ or $(q \vee (q \to p) \vee \neg p)$ is necessary and sufficient in order to prove F_1 in intuitionistic logic. For the formula $F_2 = \exists y \forall x\,(p(y) \to p(x))$, the set AxiomsHT includes three axioms (for $G \in \{p(x), \neg p(x)\}$ and $H \in \{p(y)\}$ with $G \neq H$), of which $\exists x\,(p(x) \to \forall y\, p(y))$ is necessary and sufficient to prove F_2 in intuitionistic logic.*

3.3 Intuitionistic Non-Clausal Connection Calculus

In contrast to sequent and tableau calculi, which are *connective-driven*, connection calculi [4] use a *connection*-driven search strategy, leading to a goal-oriented and, therefore, to a significantly more efficient *proof search*. A *connection* is a set $\{A_1{}^0, A_2{}^1\}$ of literals with the same predicate symbol but different polarities (in classical logic polarity 1 would represent a negated formula, representing the set $\{A_1, \neg A_2\}$).

The non-clausal connection calculus [21] works on *non-clausal* matrices, where a matrix M is a set of clauses and a clause C is a set of literals L *and* (sub)matrices. It represents a formula in negation normal form. A *prefix* is a string consisting of variables (V) and constants (a) and assigned to each literal. It can be seen as a *world path* that captures the Kripke semantics of intuitionistic logic.

For a formula F, polarity $pol \in \{0, 1\}$ and prefix p, the intuitionistic matrix $M(F^{pol} : p)$ of a prefixed formula $F^{pol} : p$ is defined inductively according to Table 1. x^* is a new term variable, t^* is the Skolem term $f^*(x_1, \ldots, x_n)$, V^* is a new prefix variable, a^* is the prefix constant of the form $f^*(x_1, \ldots, x_n)$, f^* is a new function symbol and $x_1, \ldots, x_n$ are all free term and prefix variables in the corresponding formula $F^{pol} : p$. The *intuitionistic non-clausal matrix* $M(F)$ of a formula F is the intuitionistic matrix $M(F^0 : \varepsilon)$.

Table 1: The (prefixed) non-clausal matrix for intuitionistic logic.

type	$F^{pol} : p$	$M(F^{pol} : p)$
atomic	$A^0 : p$	$\{\{A^0 : p\underline{a^*}\}\}$
α	$(G \wedge H)^1 : p$	$\{\{M(G^1 : p)\}\}, \{\{M(H^1 : p)\}\}$
	$(G \vee H)^0 : p$	$\{\{M(G^0 : p)\}\}, \{\{M(H^0 : p)\}\}$
	$(G \to H)^0 : p$	$\{\{M(G^1 : p\underline{a^*})\}\}, \{\{M(H^0 : p\underline{a^*})\}\}$
β	$(G \wedge H)^0 : p$	$\{\{M(G^0 : p), M(H^0 : p)\}\}$
	$(G \vee H)^1 : p$	$\{\{M(G^1 : p), M(H^1 : p)\}\}$
	$(G \to H)^1 : p$	$\{\{M(G^0 : p\underline{V^*}), M(H^1 : p\underline{V^*})\}\}$

type	$F^{pol} : p$	$M(F^{pol} : p)$
atomic	$A^1 : p$	$\{\{A^1 : p\underline{V^*}\}\}$
α	$(\neg G)^0 : p$	$M(G^1 : p\underline{a^*})$
	$(\neg G)^1 : p$	$M(G^0 : p\underline{V^*})$
γ	$(\forall x G)^1 : p$	$M(G[x \backslash x^*]^1 : p\underline{V^*})$
	$(\exists x G)^0 : p$	$M(G[x \backslash x^*]^0 : p)$
δ	$(\forall x G)^0 : p$	$M(G[x \backslash t^*]^0 : p\underline{a^*})$
	$(\exists x G)^1 : p$	$M(G[x \backslash t^*]^1 : p)$

As usual, a *term substitution* σ_T assigns terms to variables. Furthermore, a *prefix substitution* σ_P assigns strings to prefix variables and is calculated by a *prefix unification* [23]. For intuitionistic logic, a connection $\{A_1^0 : p_1, A_2^1 : p_2\}$ is σ*-complementary* iff $\sigma_T(A_1) = \sigma_T(A_2)$ and $\sigma_P(p_1) = \sigma_P(p_2)$ for a combined substitution $\sigma = (\sigma_T, \sigma_P)$. The *non-clausal connection calculus* for intuitionistic logic [23] is given in Figure 2. An *intuitionistic connection proof* for F is a derivation tree of $\varepsilon, M(F), \varepsilon$ in the non-clausal connection calculus, in which all leaves are axioms. Compared to the formal *clausal* connection calculus [25, 26], the extension rule is generalized and a *decomposition rule* is added [21, 22].

Example 4 *Consider the two formulas $F_3 = (p \to p)$ and $F_4 = (p \vee \neg p)$. Their matrices are $M(F_3) = \{\{p^1 : a_1V_1\}, \{p^0 : a_1a_2\}\}$ and $M(F_4) = \{\{p^0 : a_1\}, \{p^1 : a_2V_1\}\}$. The only connections in theses matrices are $\{p^1 : a_1V_1, p^0 : a_1a_2\}$ and $\{p^0 : a_1, p^1 : a_2V_1\}$, respectively. The prefixes $a_1V_1 = a_1a_2$ unify under the substitution $\sigma_P(V_1) = a_2$ whereas a_1 and a_2V_1 cannot be unified. Hence, only the first connection is σ-complementary and, therefore, F_3 is intuitionistically valid, whereas F_4 is not.*

$$\textit{Axiom (A)}\quad \frac{}{\{\},M,Path} \qquad \textit{Start (S)}\quad \frac{C_2,M,\{\}}{\varepsilon,M,\varepsilon}\ \text{and } C_2 \text{ is copy of } C_1\in M$$

$$\textit{Reduction (R)}\quad \frac{C,M,Path\cup\{L_2{:}\,\underline{p_2}\}}{C\cup\{L_1:\underline{p_1}\},M,Path\cup\{L_2:\underline{p_2}\}}\ \text{and } \{L_1{:}\underline{p_1},L_2{:}\underline{p_2}\} \text{ is } \sigma\text{-complementary}$$

$$\textit{Extension (E)}\quad \frac{C_3,M[C_1\backslash C_2],Path\cup\{L_1:\underline{p_1}\}\quad C,M,Path}{C\cup\{L_1:\underline{p_1}\},M,Path}$$

$C_3{:=}\beta\text{-}clause_{L_2}(C_2)$, C_2 is copy of C_1, C_1 is e-clause of M wrt. $Path\cup\{L_1:\underline{p_1}\}$, C_2 contains $L_2:\underline{p_2}$, $\{L_1{:}\underline{p_1},L_2{:}\underline{p_2}\}$ is σ-complementary

$$\textit{Decomposition (D)}\quad \frac{C\cup C_1,M,Path}{C\cup\{M_1\},M,Path}\ \text{and } C_1\in M_1$$

Figure 2: The non-clausal connection calculus for intuitionistic logic.

4 The Implementations

The sequent calculus LHT presented in Sections 2.2 with the optimizations described in Section 3.1 was implemented in leanHaT. The axiomatic embedding of HT presented in Section 2.3 with the optimizations described in Section 3.2 was implemented in ileanSeP-HT, ileanTAP-HT and nanoCoP-i-HT.[2]

4.1 The Sequent Prover leanHaT

leanHaT is a theorem prover for the first-order logic of HT with equality. It is a compact Prolog implementation of the sequent calculus LHT presented in Section 2.2 with the optimizations described in Section 3.1. The following (Prolog) syntax is used for formulas `F`: the connectives "`,`" (conjunction), "`;`" (disjunction), "`~`" (negation), "`=>`" (implication), "`<=>`" (equivalence), the quantifiers "`all X:F`" (universal) and "`ex X:F`" (existential); atomic formulas are represented by Prolog terms. Sequents $\Gamma\vdash\Delta$ are represented by `Left > Right` where the multisets `Left` and `Right` are represented by Prolog lists of the form `[F1,...,Fn]`. The source code of the leanHaT theorem prover is shown in Figure 3. It is divided into the specification of the sequent rules (lines *r1–r26*) and the actual proof search (lines *1–16*).

The predicate `fml(F,Pol,L1>R1,L2>R2,L3>R3,S,FreeV,NewX,Cpy1,Cpy2)` is used to specify the rules of the sequent calculus LHT in Figure 1. Each of the 22 rules of the calculus corresponds to exactly one Prolog clause of the predicate `fml`, whose number is given in parentheses after the name of the rule. Furthermore, there are four additional clauses for equivalences $A\leftrightarrow B$ (lines *r15–r18*) defined as $(A\rightarrow B)\wedge(B\rightarrow A)$. Non-splitting rules are applied first, as are quantifier rules with Eigenvariables. `F` is the main formula of the rule's conclusion, `Pol` is its polarity, which is either 0 or 1 (i.e. right or left side of the sequent), `L`n`>R`n (for $n=1,2,3$) are the formulas that are added to the n-th premise of the rule, `S` is a unique position used as Skolem functor, `FreeV` is a list of free variables in the sequent, `NewX` is the new free variable (or the empty list, otherwise), `Cpy1` is copied and unified with `Cpy2`.

The actual proof search is done by the Prolog predicate `prove(Left>Right,S,FreeV,VarLim)`. It succeeds iff there is a sequent proof for the sequent `Left>Right` with the current position `S` and the current list of free variables `FreeV` using not more than `VarLim` free variables in every proof branch.

[2] All presented theorem provers are available at `https://leancop.de/leanhat/` and `https://leancop.de/ihat/`.

```
        % - specification of sequent rules

(r1)    fml((A,B),    1, [A,B] > [],   [] > [],     [] > [], _,_,[],[],[]).
(r2)    fml((A;B),    0, [] > [A,B],   [] > [],     [] > [], _,_,[],[],[]).
(r3)    fml(~ (A,B),  0, [] > [~A,~B], [] > [],     [] > [], _,_,[],[],[]).
(r4)    fml(~ (A;B),  1, [~A,~B] > [], [] > [],     [] > [], _,_,[],[],[]).
(r5)    fml(~ (A=>B), 1, [~B] > [~A],  [] > [],     [] > [], _,_,[],[],[]).
(r6)    fml(~ (~A),   1, [] > [~A],    [] > [],     [] > [], _,_,[],[],[]).
(r7)    fml(~ (~A),   0, [~A] > [],    [] > [],     [] > [], _,_,[],[],[]).

(r8)    fml((A,B),    0, [] > [A],     [] > [B],    [] > [], _,_,[],[],[]).
(r9)    fml((A;B),    1, [A] > [],     [B] > [],    [] > [], _,_,[],[],[]).
(r10)   fml(~ (A,B),  1, [~A] > [],    [~B] > [],   [] > [], _,_,[],[],[]).
(r11)   fml(~ (A;B),  0, [] > [~A],    [] > [~B],   [] > [], _,_,[],[],[]).
(r12)   fml(~ (A=>B), 0, [~A] > [],    [] > [~B],   [] > [], _,_,[],[],[]).
(r13)   fml((A=>B),   0, [A] > [B],    [~B] > [~A], [] > [], _,_,[],[],[]).
(r14)   fml((A=>B),   1, [~A] > [],    [] > [A,~B], [B]>[],  _,_,[],[],[]).

(r15)   fml((A<=>B),  1, [((A=>B),(B=>A))] > [],[]>[],[]>[], _,_,[],[],[]).
(r16)   fml((A<=>B),  0, [] > [((A=>B),(B=>A))],[]>[],[]>[], _,_,[],[],[]).
(r17)   fml(~ (A<=>B),1, [~ ((A=>B),(B=>A))]>[],[]>[],[]>[], _,_,[],[],[]).
(r18)   fml(~ (A<=>B),0, []>[~ ((A=>B),(B=>A))],[]>[],[]>[], _,_,[],[],[]).

(r19)   fml(~all X:A, 1, [~C] > [], []>[], []>[],  S,FV,[],(X,A),(S^FV,C)).
(r20)   fml(~ex X:A,  0, [] > [~C], []>[], []>[],  S,FV,[],(X,A),(S^FV,C)).
(r21)   fml( all X:A, 0, [] > [C],  []>[], []>[],  S,FV,[],(X,A),(S^FV,C)).
(r22)   fml( ex X:A,  1, [C] > [],  []>[], []>[],  S,FV,[],(X,A),(S^FV,C)).

(r23)   fml(~all X:A, 0, [] > [~C,~all X:A], []>[],[]>[], _,_,[Y],X:A,Y:C).
(r24)   fml(~ex X:A,  1, [~C,~ex X:A] > [],  []>[],[]>[], _,_,[Y],X:A,Y:C).
(r25)   fml( all X:A  1, [C,all X:A] > [],   []>[],[]>[], _,_,[Y],X:A,Y:C).
(r26)   fml( ex X:A,  0, [] > [C,ex X:A],    []>[],[]>[], _,_,[Y],X:A,Y:C).


(a)     prove(F,I) :- prove([]>[F],s,[],I).
(b)     prove(F,I) :- \+ no_mul([([F],0)]), I1 is I+1, prove(F,I1).

        % - proof search

(1)     prove(Left>Right,_,_,_) :-
(2)         member(A,Left), ( member(B,Right) ; member(~B,Left) ), ( A==B ->
(3)         ! ; ( A=(~C) ->  C=..[P|_] ; A=..[P|_] ), P\=',', P\=';', P\='~',
(4)         P\='=>', P\='<=>', P\=all, P\=ex, unify_with_occurs_check(A,B) ).

(5)     prove(Left>Right,S,FreeV,VarLim) :-
(6)         fml(F,Pol,L1>R1,L2>R2,L3>R3,S,FreeV,V,Cpy1,Cpy2),
(7)         ( Pol=1, append(LeftA,[F|LeftB],Left), RightA=Right, RightB=[] ;
(8)           Pol=0, append(RightA,[F|RightB],Right), LeftA=Left, LeftB=[] ),
(9)         ( V=[] -> ! ; \+ length(FreeV,VarLim) ),
(10)        append(FreeV,V,FreeV1), copy_term((Cpy1,FreeV),(Cpy2,FreeV)),
(11)        append(LeftA,LeftB,L0), append(RightA,RightB,R0),
(12)        append(L1,L0,Left1), append(R1,R0,Right1), append(L2,L0,Left2),
(13)        append(R2,R0,Right2), append(L3,L0,Left3), append(R3,R0,Right3),
(14)        prove(Left1>Right1,l(S),FreeV1,VarLim),
(15)        ( L2=[],R2=[] -> true ; prove(Left2>Right2,r(S),FreeV1,VarLim) ),
(16)        ( L3=[],R3=[] -> true ; prove(Left3>Right3,x(S),FreeV1,VarLim) ).
```

Figure 3: The source code of the leanHaT sequent prover for the logic of HT.

The two axioms of the sequent calculus LHT are implemented in the first four lines (lines *1–4*). Sound unification of the two axiom literals is performed (as free variables are used). This is done by the built-in predicate `unify_with_occurs_check` (line *4*). If none of the axioms can be applied, a rule of the calculus is picked (line *6*) and checked if it can be applied, either to the left side (line *7*) or the right side (line *8*) of the current sequent. If an invertible rule is applied, any backtracking is cut off (using the Prolog cut "`!`"), otherwise it is checked whether the current number of free variables is within the current limit `VarLim` (line *9*).

Afterwards, the new free variable (if any) is added to the list of free variables and a copy of `Cpy1` is made and unified with `Cpy2` using the built-in predicate `copy_term` (line *10*). These copies are necessary whenever a quantifier rule is applied: (1) For the free variable quantifier rules $\forall$-left, $\exists$-right, $\neg\forall$-right and $\neg\exists$-left, a fresh copy of the main formula has to be added to the sequent of the premise; and (2) for the Eigenvariable quantifier rules $\forall$-right*, $\exists$-left*, $\neg\forall$-left*and $\neg\exists$-right*, a Skolem term has to be inserted for the Eigenvariable. In order to avoid that free variables are renamed, the list of free variables `FreeV` is added as an "argument" on both sides of the `copy_term` predicate. The next three lines add any new formulas R*i* and L*i* (for $i = 1,2,3$) to the right and left side of the i-th premise, respectively (lines *11–13*). Afterwards the proof search continues with the first premise (line *14*), (if existent) the second premise (line *15*) and (if existent) the third premise (line *16*).

The main predicate `prove(F,I)` realizes iterative deepening and succeeds iff there is a sequent proof for the formula `F`. The integer `I` is the (initial) limit for the number of free variables on each branch in the sequent calculus. This predicate invokes the actual proof search by calling `prove([]>[F],s,[],I)` with the start sequent `[]>[F]`, the start position `s`, the empty list `[]` of free variables and the free variable limit `I` (line *a*). If this call fails, the free variable limit is increased and the proof search starts again (line *b*). The predicate `no_mul([([F],0)])` succeeds iff there are no free variable quantifiers in `F`, i.e. quantifiers that lead to the application of a free variable quantifier rule, in which case the iterative deepening does *not* continue (line *b*) ("`\+`" is the Prolog negation). The `no_mul` predicate is implemented in six lines of code and the only code of the leanHaT (core) prover not included in Figure 3. Therefore, leanHaT is (also) a decision procedure for first-order formulas that do not contain free variable quantifiers and therefore also for the propositional fragment of HT.

If the given formula includes equality, the necessary axioms are added in a preprocessing step.

Example 5 (Sequent prover leanHaT) *Consider again the three formulas* $F_1 = (p \to q) \vee (q \to p)$, $F_2 = \exists y \forall x (p(y) \to p(x))$ *and* $F_4 = (p \vee \neg p)$. *In the ILTP problem library [31] (see also Section 5.1), these formulas are included as problems SYN416+1, SYN048+1 and SYN387+1, respectively. In order to determine whether these formulas are valid, we load the* leanHaT *prover into Prolog (using* `[leanhat].`*) and then invoke the following goals:* `prove( ( (p=>q) ; (q=>p) ), 1)`, `prove( ( ex Y: all X: ( p(Y) => p(X) ) ), 1)` *and* `prove( ( p ; ~ p ), 1)`. *For the first two formulas Prolog returns "*`Yes`*", for the third formula it returns "*`No`*". Therefore, formulas* F_1 *and* F_2 *are valid in the logic of HT, whereas* F_4 *is not.*

4.2 The Axiomatic Sequent and Tableau Provers ileanSeP-HT and ileanTAP-HT

The theorem prover ileanSeP-HT is an axiomatic extension of the intuitionistic sequent prover ileanSeP for the first-order logic of HT with equality. ileanSeP is a compact Prolog implementation of the single-succedent intuitionistic sequent calculus. Similar to tableau calculi, it uses a bottom-up proof search, free variables and a dynamic skolemization to deal with quantifiers. Together with the occurs check of term unification, this ensures that the Eigenvariable condition is respected. If necessary, equality axioms are added in a preprocessing step and treated with regard to the logic of HT.

The theorem prover ileanTAP-HT is an axiomatic extension of the intuitionistic tableau prover ileanTAP for the first-order logic of HT with equality. ileanTAP [19] is a compact Prolog implementation of a prefixed tableau calculus, similar to Fitting's prefixed tableau calculi [9]. It extends the classical calculus [32] by adding prefixes to capture the Kripke semantics of intuitionistic logic, uses free prefix variables [33] and extends skolemization to prefix constants. First, ileanTAP-HT performs a classical proof search collecting prefixes of literals that close branches. If this search succeeds, a prefix unification is used to unify these prefixes and ensure that the given formula is also intuitionistically valid. If necessary, equality axioms are added in a preprocessing step and treated with regard to HT.

Both provers for HT are likely incomplete, i.e. there are valid formulas for which they cannot find proofs, as the axiomatization is optimized with respect to the number of added axioms (see Section 3.2).

4.3 The Axiomatic Connection Prover nanoCoP-i-HT

The automated theorem prover nanoCoP-i-HT is an axiomatic extension of the intuitionistic non-clausal connection prover nanoCoP-i for the first-order logic of HT with equality. nanoCoP-i is a compact Prolog implementation of the non-clausal connection calculus for *intuitionistic* first-order logic [23, 24] and an extension of the classical prover nanoCoP [22]. It is based on the prefixed non-clausal connection calculus presented in Section 3.3. First, nanoCoP-i performs a classical proof search, in which the prefixes of each connection are collected. If this search succeeds, the prefixes of the literals in each connection are unified. Additional optimization techniques are regularity, lemmata, restricted backtracking and strategy scheduling [20, 24]. If necessary, equality axioms are added in a preprocessing step and treated with regard to the logic of HT. Currently, nanoCoP-i 2.0 is one of the fastest theorem provers for first-order intuitionistic logic [24]. Again, nanoCoP-i-HT is likely incomplete as the embedding into intuitionistic logic uses only a restricted number of axioms as specified in Section 3.2.[3]

5 Experimental Evaluation

All four implementations presented in Section 4 were evaluated on a large set of problems of an existing benchmark library and also compared to theorem provers for intuitionistic and classical logic.

5.1 Benchmark Problems

At present, larger sets of formulas for testing and benchmarking theorem provers for the first-order logic of HT are not available. So far, there exists only a small set of problems representing strong equivalences of logic programs developed in the context of the *anthem* system [8, 7]. As the syntax of HT is the same as for classical and intuitionistic logic, we can use existing benchmark formulas for these logics, even though the results on these formulas do not have any obvious meaning. As the logic of HT is an extension of intuitionistic logic, we use the ILTP problem library [31] for intuitionistic logic to test and benchmark our four theorem provers for HT. The relation between HT, intuitionistic logic and classical logic is shown in Figure 4.

[3] The focus of the presented proof calculi, methods and provers is to *prove* formulas. Effectively *refuting* formulas requires different or additional methods and techniques. For this reason, completeness (of methods and provers) is less relevant. For example, nanoCoP-i-HT proves significant more problems than leanHaT (see Section 5.2) even though it is (likely) incomplete. Furthermore, one of the most successful (proof search) techniques used in nanoCoP-i (and therefore also by nanoCoP-i-HT) is incomplete [24, 20] (overall, nanoCoP-i is complete as complete strategies are also part of the used strategy scheduling). Finally, just as first-order classical and first-order intuitionistic logic, the first-order logic of HT is also not decidable.

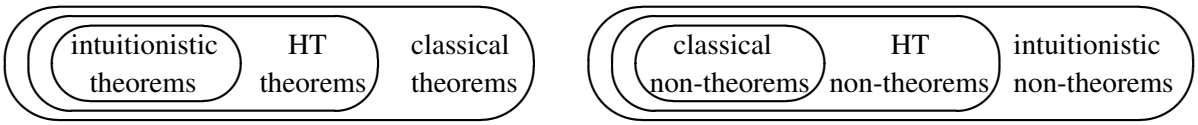


Figure 4: Relation between intuitionistic logic, HT and classical logic.

Version 1.1.2 of the ILTP problem library contains 2550 first-order formulas with status and difficulty rating information for both classical and intuitionistic logic. The problems are divided into 24 categories and taken from a wide range of different domains, such as agents (AGT), general algebra (ALG), constructive geometry (GEJ), geometry (GEO), knowledge representation (KRS), management (MGT), natural language processing (NLP), number theory (NUM), set theory (SET), software verification (SWV), syntactic (SYN) and intuitionistic syntactic (SYJ).

The problems in the ILTP library are in TPTP syntax and consist of one *Conjecture* formula and (optionally) n Axiom formulas $Axiom_1, \ldots, Axiom_n$ ($n \geq 0$). Then, the resulting formula to be proven is $(Axiom_1 \wedge \ldots \wedge Axiom_n) \rightarrow Conjecture$. If there is no *Conjecture* (formula), then the unsatisfiability of the formula $(Axiom_1 \wedge \ldots \wedge Axiom_n)$ has to be determined.

5.2 Evaluation

All four theorem provers for HT described in Section 4 were evaluated on the 2550 problems of the ILTP library. The tests were conducted on a LIFEBOOK U9311 with a 4-core i7-1185G7 CPU and 16GB of RAM, running Linux 5.15.0 and ECLiPSe Prolog version 5.10.[4]

Table 2 shows the results of the evaluation on all 2550 first-order problems of the ILTP library v1.1.2 for a CPU time limit of 10 seconds.[5] Included are the HT provers leanHaT 1.0, ileanSeP-HT 1.0, ileanTAP-HT 1.0 and nanoCoP-i-HT 2.0. Also included are the intuitionistic provers ileanSeP 1.0, ileanTAP 1.17 [19] and nanoCoP-i 2.0 [24]. The intuitionistic provers provide a lower limit for a (native) theorem prover for HT using a similar calculus. For example, a (native) sequent or tableau prover for HT should solve at least as many problems as ileanSeP, a (native) connection prover for HT at least as many problems as nanoCoP-i. The table shows for each prover the total number of proved problems, the number of problems proved within a certain time, the number of proved problems for the most important domains (see Section 5.1), the number of "unique" problems proved only by this prover (with respect to its logic), the number of refuted (invalid) problems and the number of errors (usually an overflow of the global/trail stack of ECLiPSe Prolog). As the axiomatic provers for HT are likely incomplete, they do not refute any (invalid) problems.

Of the HT provers, nanoCoP-i-HT proves by far the highest number of problems. The sequent prover leanHaT comes in second, whereas the ileanSeP and ileanTAP-based provers perform significantly worse. Both nanoCoP-i-HT and leanHaT prove the highest number of unique problems, subsuming ileanSeP-HT. By adding the conjecture start clause technique to the third strategy of nanoCoP-i-HT (i.e. using `[conj,scut,cut]` instead of `[scut,cut]`, see [24]), the number of proved problems increases to 635. Altogether, nanoCoP-i-HT and leanHaT prove a total of 652 problems.

[4] ECLiPSe Prolog 5.x is available at `https://eclipseclp.org/Distribution/Builds/`; newer versions of ECLiPSe Prolog are missing important features and have a significantly lower performance.

[5] Due to the time complexity of the underlying proof search problem, a higher time limit would only have a small impact on the number of proved problems/formulas. Only the nanoCoP-i and nanoCoP-i-HT provers would benefit a little from a higher time limit, as the strategy scheduling would then be able to execute all 12 (built-in) strategies instead of just the first nine.

Table 2: Results of the HT theorem provers on the ILTP library.

Prover	ileanTAP	ileanTAP-HT	ileanSeP	ileanSeP-HT	leanHaT	nanoCoP-i	nanoCoP-i-HT
Logic	IL	**HT**	IL	**HT**	**HT**	IL	**HT**
proved	312	154	300	209	**378**	799	**626**
0 to 1sec.	306	145	273	184	**345**	699	**528**
1 to 10sec.	6	9	27	25	**33**	100	**98**
AGT	0	0	4	3	4	15	12
ALG	6	6	0	0	3	12	9
GEJ	11	0	7	3	11	85	59
GEO	0	0	1	0	1	10	7
KRS	18	0	17	3	31	91	41
MGT	6	0	1	0	1	32	16
NLP	11	11	3	0	1	13	8
NUM	1	0	0	0	0	49	30
SET	14	1	19	12	16	112	76
SWV	51	48	82	71	107	134	127
SYN	107	56	102	93	134	125	139
SYJ	81	28	62	23	67	102	90
unique	0	3	2	0	**24**	413	**238**
refuted	4	0	4	0	**31**	88	**0**
error	60	27	151	114	**0**	3	**3**

Theorem provers for HT can also be used to refute problems in intuitionistic logic (if F is not valid in HT, then F is also not valid in intuitionistic logic). For example, some of the 18 scalable (and *classically* valid) problems SYJ207 to SYJ212 in the ILTP library are hard to refute even for *propositional* intuitionistic provers. leanHaT easily refutes all of these problems within a fraction of a second.

Table 3 shows the results of five theorem provers for classical logic: the (native) classical provers lean*TAP* 2.3 [3] and nanoCoP 2.0 [20], and the three intuitionistic provers ileanTAP, ileanSeP and nanoCoP-i using an embedding of classical into intuitionistic logic by adding the axiom schema $A \vee \neg A$ for the law of excluded middle (for all atomic formulas A). They provide an upper limit for axiomatic approaches. For example, the performance of nanoCoP-i-C is the best we can expect when using the axiomatic embedding of HT in combination with the nanoCoP-i prover.

Table 3: Results of the classical theorem provers on the ILTP library.

Prover	ileanTAP-C	ileanSeP-C	lean*TAP*	nanoCoP-i-C	nanoCoP
Logic	CL	CL	CL	CL	CL
proved	252	274	516	855	1086
0 to 1sec.	242	248	499	728	942
1 to 10sec.	10	26	17	127	144
refuted	0	0	0	5	86
error	82	105	375	3	1

6 Conclusion

Even though the logic of HT provides the logical foundation of ASP, only little effort has been done in developing tools and efficient calculi that automate formal reasoning in HT. This paper addresses this issue by introducing four automated theorem provers for the *first-order* logic of HT: leanHaT implements an optimized native sequent calculus for HT; ileanSeP-HT, ileanTAP-HT and nanoCoP-i-HT implement axiomatic embeddings into intuitionistic logic in combination with the intuitionistic theorem provers ileanSeP, ileanTAP and nanoCoP-i. To the best of our knowledge, these are the first theorem provers for the *first-order* logic of HT that do not require a translation of the given formula.

It is notable that all four presented theorem provers are based on *non-clausal* proof calculi, i.e. they do not require the translation of the given formula into a (conjunctive or disjunctive) clausal form, but preserve the structure of the original formula throughout the proof search process. This is in particular important if readable proofs are to be returned.

All four provers for HT were evaluated on the ILTP problem library providing comprehensive performance results. leanHaT proves more problems than similar sequent or tableau provers for intuitionistic logic. leanHaT solves (i.e. proves and refutes) about 20% fewer problems than the *classical* tableau prover lean*TAP*; a solid result, considering that some formulas proved by lean*TAP* are not valid in HT.

For the axiomatic embedding, a large number of axioms need to be added to the given formula. For example, for a propositional formula with 25 distinct predicate symbols, more than 1200 axioms are added, which already takes our optimized axiomatization into account. As expected and seen with similar approaches, these additional axioms increase the search space significantly. Only in combination with the intuitionistic connection prover nanoCoP-i this approach leads to a convincing performance as shown by the results of the nanoCoP-i-HT prover.

It is still an open question, which axioms are necessary for a complete and sound axiomatic embedding into intuitionistic logic. We did not find any evidence that our optimized axiomatization is incomplete for *propositional* HT. For *first-order* HT it is likely incomplete. For example, the formula $\exists y\,(\exists x\, p(x) \rightarrow p(y))$ (problem SYN971+1 in the ILTP library) can easily be proved by leanHaT, but not by any of the axiomatic theorem provers for HT. Further theoretical and practical investigations are necessary to answer this open question.

Although the performance results of leanHaT and nanoCoP-i-HT are very promising, the main focus of the research presented in this paper is not efficiency, but to initiate the development of theorem provers for the first-order logic of HT and providing a foundation for future developments of more efficient theorem provers. Neither (native) sequent or tableau calculi nor axiomatic embeddings are a basis for efficient theorem provers for HT. To this end, native proof methods that are based on a goal-oriented proof search, such as connection calculi need to be developed. This will be part of future work.

A comparison with the *anthem* system and its translation of HT into first-order classical logic will be part of future work as well. Furthermore, the development of a benchmark library for HT that also contains problems from real applications, e.g., representing strong equivalences of logic programs would be very useful and would further support the development of efficient theorem provers for HT.

Acknowledgments

The authors would like to thank the members of the *anthem* group at the University of Potsdam, the University of Nebraska at Omaha, and the University of Texas at Austin for many inspiring discussions. Thanks also goes to the anonymous reviewers for their helpful comments.

References

[1] Felicidad Aguado, Pedro Cabalar, Martin Diéguez, Gilberto Pérez & Concepción Vidal (2013): *Temporal equilibrium logic: a survey*. Journal of Applied Non-Classical Logics 23(1–2), pp. 2–24, doi:10.1080/11663081.2013.798985.

[2] Harald Beck, Minh Dao-Tran & Thomas Eiter (2016): *Equivalent Stream Reasoning Programs*. In Kambhampati [14], pp. 929–935.

[3] Bernhard Beckert & Joachim Posegga (1995): *leanTAP: Lean Tableau-based Deduction*. Journal of Automated Reasoning 15(3), pp. 339–358, doi:10.1007/BF00881804.

[4] Wolfgang Bibel (1987): *Automated Theorem Proving*. Artificial intelligence, F. Vieweg und Sohn, Wiesbaden, doi:10.1007/978-3-322-90102-6.

[5] Pedro Cabalar, Jorge Fandinno & Luis Fariñas del Cerro (2020): *Autoepistemic Answer Set Programming*. Artificial Intelligence 289, p. 103382, doi:10.1016/j.artint.2020.103382.

[6] Pedro Cabalar, Roland Kaminski, Max Ostrowski & Torsten Schaub (2016): *An ASP Semantics for Default Reasoning with Constraints*. In Kambhampati [14], pp. 1015–1021.

[7] Jorge Fandinno, Christoph Glinzer, Zachary Hansen, Jan Heuer, Yuliya Lierler, Vladimir Lifschitz, Torsten Schaub & Tobias Stolzmann (2025): *Anthem 2.0: Automated Reasoning for Answer Set Programming*. Available at `https://arxiv.org/abs/2507.11704`. (To appear in TPLP 2025).

[8] Jorge Fandinno, Vladimir Lifschitz, Patrick Lühne & Torsten Schaub (2020): *Verifying Tight Logic Programs with anthem and Vampire*. Theory and Practice of Logic Programming 20(5), pp. 735–750, doi:10.1017/S1471068420000344.

[9] Melvin Fitting (1983): *Proof Methods for Modal and Intuitionistic Logics*. D. Reidel, Dordrecht, doi:10.1007/978-94-017-2794-5.

[10] Michael Gelfond & Vladimir Lifschitz (1990): *Logic Programs with Classical Negation*. In David Warren & Péter Szeredi, editors: Proceedings of the Seventh International Conference on Logic Programming (ICLP'90), MIT Press, pp. 579–597.

[11] Gerhard Gentzen (1935): *Untersuchungen über das Logische Schließen*. Mathematische Zeitschrift 39, pp. 176–210, 405–431, doi:10.1007/BF01201353.

[12] Kurt Gödel (1932): *Zum intuitionistischen Aussagenkalkül*. Anzeiger der Akademie der Wissenschaften in Wien, pp. 65–66.

[13] Arend Heyting (1930): *Die formalen Regeln der intuitionistischen Logik*. In: Sitzungsberichte der Preussischen Akademie der Wissenschaften, Deutsche Akademie der Wissenschaften zu Berlin, pp. 42–56.

[14] Subbarao Kambhampati, editor (2016): *Proceedings of the Twenty-fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*. IJCAI/AAAI Press.

[15] Vladimir Lifschitz (2019): *Answer Set Programming*. Springer, Cham, doi:10.1007/978-3-030-24658-7.

[16] Vladimir Lifschitz, David Pearce & Agustín Valverde (2001): *Strongly equivalent logic programs*. ACM Transactions on Computational Logic 2(4), pp. 526–541, doi:10.1145/383779.383783.

[17] Vladimir Lifschitz, David Pearce & Agustín Valverde (2007): *A Characterization of Strong Equivalence for Logic Programs with Variables*. In Chitta Baral, Gerhard Brewka & John Schlipf, editors: Proceedings of the Ninth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), Lecture Notes in Artificial Intelligence 4483, Springer, Heidelberg, pp. 188–200, doi:10.1007/978-3-540-72200-7_17.

[18] Grigori Mints (2010): *Cut-free formulations for a quantified logic of here and there*. Ann. Pure Appl. Log. 162(3), pp. 237–242, doi:10.1016/J.APAL.2010.09.009.

[19] Jens Otten (1997): ileanTAP*: An Intuitionistic Theorem Prover*. In Didier Galmiche, editor: TABLEAUX 1997, Lecture Notes in Artificial Intelligence 1227, Springer, Heidelberg, pp. 307–312, doi:10.1007/BFb0027422.

[20] Jens Otten (2010): *Restricting backtracking in connection calculi*. *AI Commun.* 23(2–3), pp. 159–182, doi:10.3233/AIC-2010-0464.

[21] Jens Otten (2011): *A Non-clausal Connection Calculus*. In Kai Brünnler & George Metcalfe, editors: *TABLEAUX 2011*, *Lecture Notes in Artificial Intelligence* 6793, Springer, Heidelberg, pp. 226–241, doi:10.1007/978-3-642-22119-4_18.

[22] Jens Otten (2016): nanoCoP*: A Non-clausal Connection Prover*. In Nicola Olivetti & Ashish Tiwari, editors: *IJCAR 2016*, *Lecture Notes in Artificial Intelligence* 9706, Springer, Heidelberg, pp. 302–312, doi:10.1007/978-3-319-40229-1_21.

[23] Jens Otten (2017): *Non-clausal Connection Calculi for Non-classical Logics*. In Renate Schmidt & Cláudia Nalon, editors: *TABLEAUX 2017*, *Lecture Notes in Artificial Intelligence* 10501, Springer, Cham, pp. 209–227, doi:10.1007/978-3-319-66902-1_13.

[24] Jens Otten (2021): *The* nanoCoP *2.0 Connection Provers for Classical, Intuitionistic and Modal Logics*. In Anupam Das & Sara Negri, editors: *TABLEAUX 2021*, *Lecture Notes in Artificial Intelligence* 12842, Springer, Cham, pp. 236–249, doi:10.1007/978-3-030-86059-2_14.

[25] Jens Otten & Wolfgang Bibel (2003): leanCoP*: lean connection-based theorem proving*. *Journal of Symbolic Computation* 36(1–2), pp. 139–161, doi:10.1016/S0747-7171(03)00037-3.

[26] Jens Otten & Wolfgang Bibel (2017): *Advances in Connection-Based Automated Theorem Proving*. In Mike Hinchey, Jonathan P. Bowen & Ernst-Rüdiger Olderog, editors: *Provably Correct Systems*, NASA Monographs in Systems and Software Engineering, Springer, Cham, pp. 211–241, doi:10.1007/978-3-319-48628-4_9.

[27] David Pearce (2006): *Equilibrium logic*. *Annals of Mathematics and Artificial Intelligence* 47(1–2), pp. 3–41, doi:10.1007/s10472-006-9028-z.

[28] David Pearce, Inmaculada de Guzmán & Augustín Valverde (2000): *A Tableau Calculus for Equilibrium Entailment*. In Roy Dyckhoff, editor: *Proceedings of the Ninth International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 2000)*, *Lecture Notes in Computer Science* 1847, Springer, Heidelberg, pp. 352–367, doi:10.1007/10722086_28.

[29] David Pearce & Augustín Valverde (2005): *A First Order Nonmonotonic Extension of Constructive Logic*. *Studia Logica* 30(2–3), pp. 321–346, doi:10.1007/s11225-005-8473-8.

[30] Francis Jeffry Pelletier (1986): *Seventy-Five Problems for Testing Automatic Theorem Provers*. *Journal of Automated Reasoning* 2(2), pp. 191–216, doi:10.1007/BF02432151.

[31] Thomas Raths, Jens Otten & Christoph Kreitz (2007): *The ILTP problem library for intuitionistic logic*. *Journal of Automated Reasoning* 38, pp. 261–271, doi:10.1007/s10817-006-9060-z.

[32] Raymond M. Smullyan (1968): *First-Order Logic*. Ergebnisse der Mathematik und ihrer Grenzgebiete, Springer, Berlin, Heidelberg, New York, doi:10.1007/978-3-642-86718-7.

[33] Lincoln A. Wallen (1990): *Automated Deduction in Nonclassical Logics*. MIT Press, Cambridge, Mass.

Automated Theorem Proving for Prolog Verification*

Fred Mesnard Thierry Marianne

Étienne Payet

LIM, université de La Réunion, France

{frederic.mesnard,thierry.marianne,etienne.payet}@univ-reunion.fr

LPTP (Logic Program Theorem Prover) is an interactive natural-deduction-based theorem prover for pure Prolog programs with negation as failure, unification with the occurs check, and a restricted but extensible set of built-in predicates. With LPTP, one can formally prove termination and partial correctness of such Prolog programs. LPTP was designed in the mid-1990's by Robert F. Stärk. It is written in ISO-Prolog and comes with an Emacs user-interface.

From a theoretical point of view, in his publications about LPTP, Stärk associates a set of first-order axioms IND(P) to the considered Prolog program P. IND(P) contains the Clark's equality theory for P, definitions of success, failure and termination for each user-defined logic procedure in P, axioms relating these three points of view, and an axiom schema for proving inductive properties. LPTP is thus a dedicated proof editor where these axioms are hard-wired.

We propose to translate these axioms as first-order formulas (FOFs), and apply automated theorem provers to check the property of interest. Using FOF as an intermediary language, we experiment the use of automated theorem provers for Prolog program verification. We evaluate the approach over a benchmark of about 400 properties of Prolog programs from the library available with LPTP. Both the compiler which generates a set of FOF files from a given input Prolog program together with its properties and the benchmark are publicly available.

1 Introduction

In the mid-1990's, Robert F. Stärk defined a framework for Prolog verification [26, 29]. He considered a subset of ISO-Prolog [13]: pure Prolog programs with negation as failure, unification with the occurs check, and allowed a restricted but extensible set of built-in predicates. He presented a first-order formalisation with axiom schemas of the usual operational semantics of Prolog. A safeness condition included in termination condition imposes groundness before evaluation of negated goals. He showed soundness and completeness for termination, success, and failure. The framework also allows partial correctness properties to be proved by induction w.r.t. the clauses defining predicates, considered as inductive definitions. Some examples will be discussed in Section 3 and Section 4. The logical theory was hard-wired in an interactive dedicated first-order natural-deduction-based theorem prover called LPTP (Logic Program Theorem Prover). Stärk implemented LPTP in ISO-Prolog, together with an Emacs user-interface, an HTML and TEX manager, a detailed user-manual, and a library of predicates for Peano numbers, integers, lists, sorting algorithms, etc. with numerous proven properties. A copy of LPTP is vailable at `https://github.com/FredMesnard/lptp`.

Thirty years later, LPTP is still running on any ISO-Prolog processor, with its initial interface. Today, formal verification of computer programs is an established discipline within computer science. Nonetheless, program verification by interactive theorem proving is still a slow process and requires non-trivial

*This paper is an updated version of [21] that appeared in the 2024 LPAR Complementary Volume.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 469–481, doi:10.4204/EPTCS.439.32

© F. Mesnard, T. Marianne & É. Payet
This work is licensed under the
Creative Commons Attribution License.

skills. On the other hand, during the last three decades, the increase in computing power and the advances in automated theorem proving have been notable. For instance, TPTP (Thousands of Problems for Theorem Provers, [30]) is a library of test problems for automated theorem proving. It provides online tools to check the syntax of input problems and apply a bunch of user selected automated theorem provers. Among them, E [25] and Vampire [16] are two powerful freely available automated theorem provers, performing very well in many international competitions over the years. Interactive theorem prover implementers were able to take advantage of these progress by implementing so-called *hammers* for their tools, see e.g. [23, 3].

This evolution raises the following questions: can we also use the TPTP FOF *Esperanto* to formulate the logic theory Stärk associates to a logic program? Can we use *off-the-shelf* TPTP provers and obtain automatic proofs in reasonable time? Can we get an acceptable success rate with such an approach?

The main contribution of this paper is the following. Using FOF (*First-Order Form*, one of the logic languages proposed by TPTP, see [31]) as an intermediary language, we describe the first – to the best of our knowledge – experiment of the use of automated theorem provers, namely E and Vampire, for Prolog program verification, including termination and partial correctness. We evaluate the approach over about 400 properties of Prolog programs. Both the compiler applying Stärk's theory to a given input Prolog program and its properties to a set of FOF files and the benchmark are publicly available at `https://github.com/atp-lptp/automated-theorem-proving-for-prolog-verification`.

We organize the paper as follows. The next section presents a brief summary of the LPTP system. The third section describes step by step how to compile a Prolog program, its associated LPTP axioms and a property of interest into a FOF file. Then we present an experimental evaluation, related work and we conclude.

2 Notation

FOF (*First Order Form*) is a well-known logic language from TPTP for expressing first-order logic (FOL) axioms and conjectures. A formula is written `fof(`*name*`,`*role*`,`*formula*`)`, where *name* is the name of the formula, *role* is either `axiom` or `conjecture` and *formula* is informally defined as follows:

FOL	FOF	FOL	FOF
$A \wedge B$	`A & B`	$\neg p(x)$	`~ p(X)`
$A \vee B$	`A | B`	$\exists x.A$	`?[X] : A`
$A \rightarrow B$	`A => B`	$\forall x.A$	`![X] : A`

Numerous examples will appear in the next sections.

Let P be a pure logic program where negative literals may appear in the body of clauses (also called *normal program* in [19]). For sake of conciseness, we do not consider built-in predicates (see [29] for a full treatment) other than the equality `=/2`. We start with $\mathscr{L}$, the first-order language associated with P. The *goals* of $\mathscr{L}$ are:

$$G,H ::= \texttt{true} \mid \texttt{fail} \mid s = t \mid A \mid \texttt{\textbackslash+}\ G \mid (G,H) \mid (G;H) \mid \texttt{some}\ x\ G$$

where s and t are two terms, x is a variable and A is an atomic goal. The goals of $\mathscr{L}$ have the operational semantics specified by ISO-Prolog [13] assuming the occurs check.

$\hat{\mathscr{L}}$ is the specification language of LPTP. For each user-defined predicate symbol R, $\hat{\mathscr{L}}$ does not include R, but instead it contains three predicate symbols R^s, R^f, R^t of the same arity as R, which respectively express success, failure and termination of R. $\hat{\mathscr{L}}$ also contains a unary constraint for groundness

gr, expressing that its argument is ground. The *formulas* of $\hat{\mathscr{L}}$ are:

$$\phi, \psi ::= \top \mid \bot \mid s = t \mid R(\vec{t}) \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x\phi \mid \exists x\phi$$

where $\vec{t}$ is a sequence of n terms and R denotes a n-ary predicate symbol of $\hat{\mathscr{L}}$. The semantics of $\hat{\mathscr{L}}$ is the first-order predicate calculus of classical logic.

For any of the user-defined logic procedure R in a logic program P, $D_R^P(\vec{x})$ denotes its Clark's *if-and-only-if* completed definition, cf. [5, 19].

For defining the declarative semantics of logic programs, Stärk uses three syntactic operators **S**, **F** and **T** which map goals of $\mathscr{L}$ into $\hat{\mathscr{L}}$-formulas. Intuitively, **S***G* means *G* succeeds (any breadth-first evaluation of *G* succeeds), **F***G* means *G* fails (the ISO-Prolog evaluation stops without any answer), and **T***G* means *G* terminates (the ISO-Prolog evaluation produces a finite number of answers then stops). The definition of the operators follows:

$\mathbf{S}R(\vec{t}) := R^s(\vec{t})$ $\quad$ $\mathbf{S}\,\texttt{true} := \top$ $\quad$ $\mathbf{S}\,\texttt{fail} := \bot$ $\quad$ $\mathbf{S}(s=t) := (s=t)$
$\mathbf{S}\backslash\!+G := \mathbf{F}G$ $\quad$ $\mathbf{S}(G,H) := \mathbf{S}G \wedge \mathbf{S}H$ $\quad$ $\mathbf{S}(G;H) := \mathbf{S}G \vee \mathbf{S}H$ $\quad$ $\mathbf{S}(\texttt{some}\ x\ G) := \exists x \mathbf{S}G$

$\mathbf{F}R(\vec{t}) := R^f(\vec{t})$ $\quad$ $\mathbf{F}\,\texttt{true} := \bot$ $\quad$ $\mathbf{F}\,\texttt{fail} := \top$ $\quad$ $\mathbf{F}(s=t) := \neg(s=t)$
$\mathbf{F}\backslash\!+G := \mathbf{S}G$ $\quad$ $\mathbf{F}(G,H) := \mathbf{F}G \vee \mathbf{F}H$ $\quad$ $\mathbf{F}(G;H) := \mathbf{F}G \wedge \mathbf{F}H$ $\quad$ $\mathbf{F}(\texttt{some}\ x\ G) := \forall x \mathbf{F}G$

$\mathbf{T}R(\vec{t}) := R^t(\vec{t})$ $\quad$ $\mathbf{T}\,\texttt{true} := \top$
$\mathbf{T}\,\texttt{fail} := \top$ $\quad$ $\mathbf{T}(s=t) := \top$
$\mathbf{T}\backslash\!+G := \mathbf{T}G \wedge gr(G)$ $\quad$ $\mathbf{T}(G,H) := \mathbf{T}G \wedge (\mathbf{F}G \vee \mathbf{T}H)$
$\mathbf{T}(G;H) := \mathbf{T}G \wedge \mathbf{T}H$ $\quad$ $\mathbf{T}(\texttt{some}\ x\ G) := \forall x \mathbf{T}G$

Note that termination requires a safe use of negation, see the definition of $\mathbf{T}\backslash\!+G$ where the goal *G* has to be proved terminating and ground at proof time. Finally, we add the definition of *gr*, which belongs to the specification language and is needed for defining $\mathbf{T}\backslash\!+G$:

$gr(\texttt{true}) := \top$ $\quad$ $gr((G,H)) := gr(G) \wedge gr(H)$
$gr(\texttt{fail}) := \top$ $\quad$ $gr((G;H)) := gr(G) \wedge gr(H)$
$gr(s=t) := gr(s) \wedge gr(t)$ $\quad$ $gr(\texttt{some}\ x\ G) := \exists x\ gr(G)$
$gr(R(t_1,\ldots,t_n)) := gr(t_1) \wedge \ldots \wedge gr(t_n)$ $\quad$ $gr(\backslash\!+G) := gr(G)$

We refer the reader to the papers of Stärk [26, 27, 28, 29] for a complete presentation of LPTP.

3 Compiling LPTP axioms to FOF

With LPTP, we prove properties of a logic program *P* w.r.t. its *inductive extension* IND(*P*) which includes Clark's completion [5] and induction along the definition of the predicates. Stärk shows that the inductive extension is always consistent and proves various correctness and completeness results w.r.t. the operational semantics of Prolog [29]. The first-order theory IND(*P*) (cf. [29], pp. 253–254) is defined by nine axiom schemas which we describe now, along with their translation in FOF. We omit the fixed point axioms for builtins. Let us also point out that the specification language $\hat{\mathscr{L}}$ of LPTP can be extended by new function and predicate symbols at a logical level (there is no associated Prolog code). As shown in [21], such function and predicate definitions can also be compiled into FOF.

3.1 First steps

Let us consider the following logic program ADD as our running example.

```
nat(0).                          add(0,Y,Y).
nat(s(X)) :- nat(X).             add(s(X),Y,s(Z)) :- add(X,Y,Z).
```

We discuss the axioms proposed by Stärk and apply them to the ADD program.

The axioms of Clark's equality theory

1. $f(x_1,\ldots,x_n) = f(y_1,\ldots,y_n) \rightarrow x_i = y_i$ [if f is n-ary and $1 \leq i \leq n$]
2. $f(x_1,\ldots,x_n) \neq g(y_1,\ldots,y_m)$ [if $n \neq m$ or $f \not\equiv g$]
3. $t \neq x$ [if x occurs in t and $t \not\equiv x$]

The first two axioms specify some properties of the trees built from the function symbols extracted from the program under consideration. The third axiom forbids infinite trees. Note that it is actually an axiom schema, i.e., an infinite set of first order axioms. We will omit it but we stay sound. Here is the FOF version (see Section 2 for the FOF syntax):

```
fof(id1,axiom,! [Xx4] : ! [Xx5] : (s(Xx4) = s(Xx5) => Xx4 = Xx5)).
fof(id2,axiom,! [Xx3] : ~ ('0' = s(Xx3))).
```

Axioms for gr/1

4. $\mathrm{gr}(c)$ [if c is a constant]
5. $\mathrm{gr}(x_1) \wedge \ldots \wedge \mathrm{gr}(x_m) \leftrightarrow \mathrm{gr}(f(x_1,\ldots,x_m))$ [f is m-ary]

Actually, LPTP deals with *non-ground* terms and offers a predefined predicate $gr/1$ that we can consider as a constraint. This relation is useful for instance for dealing with negation as failure as LPTP only allows negation by failure for *ground* goals (see the definition $\mathbf{T}\backslash{+}G$). Back to our example, here is the FOF version:

```
fof(id4,axiom,gr('0')).
fof(id5,axiom,! [Xx6] : (gr(Xx6) <=> gr(s(Xx6)))).
```

The ADD program contains two user-defined predicates, `add/3` and `nat/1`. LPTP considers each user-defined predicate through three points of view: failure, success and termination. So LPTP creates the following predicates: `add_fails/3`, `add_succeeds/3`, `add_terminates/3`, and similarly for `nat/1`. These three viewpoints are linked with the following axioms, where R^s (resp. R^f and R^t) denotes `R_succeeds/3` (resp. `R_fails/3` and `R_terminates/3`).

Uniqueness axioms and totality axioms

6. $\neg(R^s(\vec{x}) \wedge R^f(\vec{x}))$ [if R is a user-defined predicate]
7. $R^t(\vec{x}) \rightarrow (R^s(\vec{x}) \vee R^f(\vec{x}))$ [if R is a user-defined predicate]

Axiom 6 says that for any tuple of (possibly non-ground) terms, we cannot have at the same time success and failure of R. Axiom 7 states that given termination, we have success or failure. Altogether, it means that for any tuple of terms $\vec{x}$, assuming termination, either $R(\vec{x})$ succeeds or (exclusively) $R(\vec{x})$ fails. So for our example, we get:

```
fof(ida6,axiom,! [Xx7,Xx8,Xx9] :
```

```
    ~ ((add_succeeds(Xx7,Xx8,Xx9) & add_fails(Xx7,Xx8,Xx9)))).
fof(ida7,axiom,! [Xx7,Xx8,Xx9] :
    (add_terminates(Xx7,Xx8,Xx9) =>
    (add_succeeds(Xx7,Xx8,Xx9) | add_fails(Xx7,Xx8,Xx9)))).
fof(idn6,axiom,! [Xx10] :
    ~ ((nat_succeeds(Xx10) & nat_fails(Xx10)))).
fof(idn7,axiom,! [Xx10] :
    (nat_terminates(Xx10) =>
    (nat_succeeds(Xx10) | nat_fails(Xx10)))).
```

Fixed point axioms for user-defined predicates R

8. $R^s(\vec{x}) \leftrightarrow \mathbf{S}D_R^P(\vec{x})$, $R^f(\vec{x}) \leftrightarrow \mathbf{F}D_R^P(\vec{x})$, $R^t(\vec{x}) \leftrightarrow \mathbf{T}D_R^P(\vec{x})$

We recall that $D_R^P(\vec{x})$ denotes the definition of the completion [5] of the user-defined procedure $R(\vec{x})$ in the logic program P. In the previous section, we saw how to apply the operator **S**, **F** and **T** to formulas. So for instance, the first equivalence $R^s(\vec{x}) \leftrightarrow \mathbf{S}D_R^P(\vec{x})$ defines $R^s(\vec{x})$. Back to our running example, we get:

```
fof(idns8,axiom,! [Xx1] : (nat_succeeds(Xx1) <=>
    (? [Xx2] : (Xx1 = s(Xx2) & nat_succeeds(Xx2)) | Xx1 = '0'))).
fof(idnf8,axiom,! [Xx1] : (nat_fails(Xx1) <=>
    (! [Xx2] : (~ (Xx1 = s(Xx2)) |
                nat_fails(Xx2)) & ~ (Xx1 = '0')))).
fof(idnt8,axiom,! [Xx1] : (nat_terminates(Xx1) <=>
    (! [Xx2] : ((~ (Xx1 = s(Xx2)) | nat_terminates(Xx2)))))).
```

and similarly for `add/3`.

Finally, for any property of the form $\forall\vec{x}[R^s(\vec{x}) \rightarrow \phi(\vec{x})]$, where $R(\vec{x})$ is a user-defined procedure and $\phi(\vec{x})$ an $\hat{\mathscr{L}}$ -formula, we have an induction schema. The interactive prover LPTP is able to *dynamically* generate an induction axiom on demand while the user interacts with it. In our approach, we *statically* generate the induction axiom *once* from the conjecture to be proved, if the conjecture can be easily rewritten as required. This is a potential source of imprecision, but again we stay sound. Let us examine a simple case. It is exactly what happens using LPTP, which slightly generalizes [29]. By *directly recursive user-defined predicate* in the box below, we forbid mutual recursive definitions. Of course, LPTP is able to handle mutually recursive properties, see [26] for some examples.

A (simplified) induction schema for a user-defined predicate R

Let R be a directly recursive user-defined predicate and let $\phi(\vec{x})$ be an $\hat{\mathscr{L}}$-formula such that the length of $\vec{x}$ is equal to the arity of R.
Let $sub(\phi(\vec{x})/R)$ be the formula to be proven $\forall\vec{x}(R^s(\vec{x}) \rightarrow \phi(\vec{x}))$.
Let $closed(\phi(\vec{x})/R)$ be the formula obtained from $\forall\vec{x}(\mathbf{S}D_R^P(\vec{x}) \rightarrow R^s(\vec{x}))$ by replacing

- $R^s(\vec{x})$ by $\phi(\vec{x})$ on the right of $\rightarrow$,
- all occurrences of $R(\vec{t})$ appearing on the left of $\rightarrow$ by $\phi(\vec{t}) \wedge R(\vec{t})$.

Then the induction axiom is the following formula:

9. $closed(\phi(\vec{x})/R) \rightarrow sub(\phi(\vec{x})/R)$

Let us apply this axiom to the following property, informally stated as: for any term x, if `nat(`x`)` then `add(`x`,0,`x`)`. Expressed in LPTP, it gives: for any term x, if `nat_succeeds(`x`)` then `add_succeeds(`x`,0,`x`)`, which is exactly the formula $sub(\phi(\vec{x})/R)$ of axiom 9. So $R \equiv$ `nat`, $R^s \equiv$ `nat_succeeds` and $\phi(\vec{x}) \equiv$ `add_succeeds`$(x,0,x)$.

For the left-hand side of axiom 9, we start from

$$\forall x(\mathbf{S}D_{nat}^{ADD}(x) \rightarrow \texttt{nat_succeeds}(x))$$

We have $D_{nat}^{ADD}(x) \equiv x = 0 \vee \exists y(x = s(y) \wedge \texttt{nat}(y))$. We replace $\texttt{nat}(y)$ by $\texttt{nat}(y) \wedge \texttt{add_succeeds}(y,0,y)$. We replace $\texttt{nat_succeeds}(x)$ by $\texttt{add_succeeds}(x,0,x)$. We get: $\forall x\ (\mathbf{S}\ [x = 0 \vee \exists y(x = s(y) \wedge \texttt{nat}(y) \wedge \texttt{add_succeeds}(y,0,y))] \rightarrow \texttt{add_succeeds}(x,0,x))$. We apply $\mathbf{S}$ and obtain: $\forall x([x = 0 \vee \exists y(x = s(y) \wedge \texttt{nat_succeeds}(y) \wedge \texttt{add_succeeds}(y,0,y))] \rightarrow \texttt{add_succeeds}(x,0,x))$.

Summarizing, in FOF, associated with the property to be proved:

```
fof(lemma,conjecture,
! [Xx] : (nat_succeeds(Xx) => add_succeeds(Xx,'0',Xx))).
```

we obtain the following induction axiom:

```
fof(induction,axiom,(
    ! [Xx] :
        ((? [Xx2] : (Xx = s(Xx2) & (nat_succeeds(Xx2)
                                    & add_succeeds(Xx2,'0',Xx2)))
        | Xx = '0') => add_succeeds(Xx,'0',Xx))
=>
    ! [Xx] : (nat_succeeds(Xx) => add_succeeds(Xx,'0',Xx)))).
```

We can gather all the 15 axioms, including the axioms defining `add_success/3`, `add_fails/3`, and `add_terminates/3` and the conjecture plus its induction axiom in a file, say `test.fof` and submit it to the E prover or to Vampire. Both systems will find a refutation in a fraction of a second on a standard laptop.

It allows us to conclude for any term x, if `nat(`x`)` then `add(`x`,0,`x`)` is true. Operationally, for any natural number n, in the Prolog search tree corresponding to the goal $\texttt{add}(s^n(0),0,s^n(0))$, the empty clause appears. Assuming termination, which will be shown later, it means that the user will get (at least) one positive answer for the query `:-` $\texttt{add}(s^n(0),0,s^n(0))$. when executed with any ISO-Prolog system.

Here's the manual proof of the same property in its LPTP version (a Prolog file), followed by its TEX version produced by LPTP. Using the interactive LPTP Emacs mode, we began this proof by invoking the

ind tactic, asking for an inductive proof. Both the base case and the inductive case were automatically generated and completed by LPTP.

```
:- lemma(add:x_0_x, all [x]: succeeds nat(?x) => succeeds add(?x,0,?x),
induction([all x: succeeds nat(?x) => succeeds add(?x,0,?x)],
 [step([],[],[],succeeds add(0,0,0)),
  step([x], [succeeds add(?x,0,?x), succeeds nat(?x)], [],
   succeeds add(s(?x),0,s(?x)))])).
```

Lemma [*add:x_0_x*] $\forall x\,(\mathbf{S}\,\mathtt{nat}(x) \rightarrow \mathbf{S}\,\mathtt{add}(x,0,x))$.

Proof.

Induction$_0$: $\forall x\,(\mathbf{S}\,\mathtt{nat}(x) \rightarrow \mathbf{S}\,\mathtt{add}(x,0,x))$.

Hypothesis$_1$: none.

Conclusion$_1$: $\mathbf{S}\,\mathtt{add}(0,0,0)$.

Hypothesis$_1$: $\mathbf{S}\,\mathtt{add}(x,0,x)$ and $\mathbf{S}\,\mathtt{nat}(x)$.

Conclusion$_1$: $\mathbf{S}\,\mathtt{add}(\mathtt{s}(x),0,\mathtt{s}(x))$. □

3.2 A second property

Now let us consider the following property: for any x, y and z such that $\mathtt{nat}(x)$, $\mathtt{nat}(y)$ and $\mathtt{add}(s(x),y,z)$, we have $\mathtt{add}(x,s(y),z)$. Let us first assert the previous property as an axiom, which can now be freely used by the automated prover, then we have our new conjecture:

```
fof('lemma-(add:x_0_x)',axiom,
  ! [Xx] : (nat_succeeds(Xx) => add_succeeds(Xx,'0',Xx))).

fof('lemma-(add:succ)',conjecture,
  ! [Xx,Xy,Xz] : (((nat_succeeds(Xx) & nat_succeeds(Xy))
                   & add_succeeds(s(Xx),Xy,Xz))
                   => add_succeeds(Xx,s(Xy),Xz))).
```

In order to generate an induction axiom for this property, we first rewrite it in the form $\forall \vec{x}[R^s(\vec{x}) \rightarrow \phi(\vec{x})]$ and we apply the simplified induction schema for user-defined predicates. It gives:

```
fof(induction,axiom,(
! [Xx] :
    ((? [Xy25] :
      (Xx = s(Xy25) & (nat_succeeds(Xy25)
       & ! [Xy,Xz] : ((add_succeeds(s(Xy25),Xy,Xz)
                       & nat_succeeds(Xy))
      => add_succeeds(Xy25,s(Xy),Xz))))
    | Xx = '0') =>
      ! [Xy,Xz] : ((add_succeeds(s(Xx),Xy,Xz) & nat_succeeds(Xy))
         => add_succeeds(Xx,s(Xy),Xz)))
 => ! [Xx] : (nat_succeeds(Xx)
    => ! [Xy,Xz] : ((add_succeeds(s(Xx),Xy,Xz) & nat_succeeds(Xy))
                    => add_succeeds(Xx,s(Xy),Xz))))).
```

Again, we can gather all axioms, the conjecture and its induction axiom in a file and submit it to Vampire, which will find a refutation in about one minute.

3.3 Commutativity of Peano addition

We are now equipped to consider commutativity of Peano addition: for any x,y,z, if $\texttt{add}(x,y,z)$ then $\texttt{add}(y,x,z)$. Of course, stated this way, the property is false. We need to enforce that x and y are Peano numbers. So first we add our two previous properties as axioms. Here is our new conjecture, associated with its induction axiom:

```
fof('theorem-(add:commutative)',conjecture,
  ! [Xx,Xy,Xz] : (((nat_succeeds(Xx) & nat_succeeds(Xy))
                   & add_succeeds(Xx,Xy,Xz))
                  => add_succeeds(Xy,Xx,Xz))).

fof(induction,axiom,
(! [Xx] :
  ((? [Xy26] : (Xx = s(Xy26) & (nat_succeeds(Xy26)
    & ! [Xy,Xz] : ((add_succeeds(Xy26,Xy,Xz) & nat_succeeds(Xy))
    => add_succeeds(Xy,Xy26,Xz))))
    | Xx = '0') =>
      ! [Xy,Xz] : ((add_succeeds(Xx,Xy,Xz) & nat_succeeds(Xy))
    => add_succeeds(Xy,Xx,Xz)))
=>
! [Xx] : (nat_succeeds(Xx) =>
  ! [Xy,Xz] : ((add_succeeds(Xx,Xy,Xz) & nat_succeeds(Xy))
  => add_succeeds(Xy,Xx,Xz))))).
```

The conjecture is proved in a fraction of a second by Vampire.

3.4 Some termination proofs

Finally, let us prove some termination properties about `add/3`. It is immediate to see that the Prolog proof of $\texttt{add}(x,y,z)$ terminates if $\texttt{nat}(x)$ or $\texttt{nat}(z)$. We prove this by stating two lemmas which we will gather in a theorem. Here are the LPTP properties and their proofs (we omit the second one).

Lemma [*add:term:1*] $\forall x,y,z\,(\mathbf{S}\,\texttt{nat}(x) \rightarrow \mathbf{T}\,\texttt{add}(x,y,z))$. **Proof.**

Induction$_0$: $\forall x\,(\mathbf{S}\,\texttt{nat}(x) \rightarrow \forall y,z\,\mathbf{T}\,\texttt{add}(x,y,z))$.

Hypothesis$_1$: none.

Conclusion$_1$: $\forall y,z\,\mathbf{T}\,\texttt{add}(0,y,z)$.

Hypothesis$_1$: $\forall y,z\,\mathbf{T}\,\texttt{add}(x,y,z)$ and $\mathbf{S}\,\texttt{nat}(x)$.

Conclusion$_1$: $\forall y,z\,\mathbf{T}\,\texttt{add}(\texttt{s}(x),y,z)$. □

Lemma [*add:term:3*] $\forall x,y,z\,(\mathbf{S}\,\texttt{nat}(z) \rightarrow \mathbf{T}\,\texttt{add}(x,y,z))$. **Proof.** Similar. □

Theorem [*add:term*] $\forall x,y,z\,(\mathbf{S}\,\mathtt{nat}(x) \vee \mathbf{S}\,\mathtt{nat}(z) \rightarrow \mathbf{T}\,\mathtt{add}(x,y,z))$. **Proof.**

Assumption$_0$: $\mathbf{S}\,\mathtt{nat}(x) \vee \mathbf{S}\,\mathtt{nat}(z)$.

Case$_1$: $\mathbf{S}\,\mathtt{nat}(x)$. $\mathbf{T}\,\mathtt{add}(x,y,z)$ by Lemma 1 [*add:term:1*].

Case$_1$: $\mathbf{S}\,\mathtt{nat}(z)$. $\mathbf{T}\,\mathtt{add}(x,y,z)$ by Lemma 2 [*add:term:3*].

Hence$_1$, in all cases: $\mathbf{T}\,\mathtt{add}(x,y,z)$.

Thus$_0$: $\mathbf{S}\,\mathtt{nat}(x) \vee \mathbf{S}\,\mathtt{nat}(z) \rightarrow \mathbf{T}\,\mathtt{add}(x,y,z)$. □

Each of the three statements is proved in a fraction of a second by Vampire. Our compiler generates an instance of the induction axiom for each lemma and not for the theorem. For instance, here is the first conjecture and its induction axiom:

```
fof('lemma-(add:term:1)',conjecture,
  ! [Xx,Xy,Xz] : (nat_succeeds(Xx) => add_terminates(Xx,Xy,Xz))).

fof(induction,axiom,(
! [Xx] :
  ((? [Xx2] : (Xx = s(Xx2) & (nat_succeeds(Xx2)
                     & ! [Xy,Xz] : add_terminates(Xx2,Xy,Xz)))
    | Xx = '0')
  => ! [Xy,Xz] : add_terminates(Xx,Xy,Xz))
=>
! [Xx] : (nat_succeeds(Xx) => ! [Xy,Xz] : add_terminates(Xx,Xy,Xz)))).
```

4 Experimental Results

We applied the schema explained in the previous sections to various libraries available with LPTP which we summarize now. The library `nat` defines some basic Peano relations with the expected properties. The library `ack` defines the relational version of the Ackermann function with three properties (see below). The library `gcd` defines a version of the greatest common divisor relation, with its full correctness proof. The library `int` defines integers. The library `list` proposes some elementary relations about lists with their properties. The library `suffix` defines two versions of the sublist relation, one as the prefix of a suffix, the other as the suffix of a prefix, and shows that the two versions are equivalent w.r.t. termination, success and failure. Similarly, the library `reverse` defines the two classical versions of the reverse relation, one with the append relation, the other with an accumulator and shows their full equivalence. The library `permutation` defines the permutation relation with some useful properties for the correctness proofs of the sorting algorithms defined in the libraries `sort` and `mergesort`. The library `taut` defines a tautology checker for propositional formulas, together with its correctness proof (see [27] for a detailed description).

How do we process such files? Given a program from the LPTP library, we first enumerate the requirements for trying to prove the properties listed in its associated LPTP proof file. Requirements are the logic program P and the associated LPTP proof file. If P depends on other logic programs, we must include them. If the associated LPTP proof file uses other proof files, we must include them as well. We assume there is no circularity such as assuming a lemma before trying to prove it. We use these requirements to build a target logic program P' and a target LPTP proof file. Then P' is compiled into the FOF version of IND(P'). Each fact (i.e., lemma, corollary or theorem) is compiled as a FOF conjecture (possibly with its induction axiom) and stored in a single file. Such file also contains the logic theory

IND(P') compiled as FOF axioms. Previously processed FOF conjectures are converted as FOF axioms as well. As a result, we produce as many FOF files as there are facts in the initial LPTP proof file. At last, both the E Theorem Prover and Vampire are applied to each FOF file with the commands `vampire -mode casc -m 16384 -cores 7 -t $TO $FILE` and `eprover -delete-bad-limit=2000000000 -definitional-cnf -s -auto-schedule=8 -proof-object -cpu-limit=$TO $FILE`.

We gather the results in Table 1. The first column gives the library names. The second column gives the number of (lemmas/corollaries/theorems) of the associated proof file. The remaining nine columns can be divided in three groups. On a MacBook Pro, 8 cores, M2, 24 GB, macOS Sonoma, the first group gives the success rate for a 1 second timeout for the E prover (column E-1s), Vampire (column V-1s) and for the combination of the two provers (column EV-1s). The second group (resp. third group) gives the success rate for a timeout of 10 seconds (resp. 60 seconds).

lib	#	E-1s	V-1s	EV-1s	E-10s	V-10s	EV-10s	E-60s	V-60s	EV-60s
nat	91	70%	88%	88%	76%	95%	95%	78%	97%	97%
gcd	11	45%	45%	45%	45%	45%	45%	45%	45%	45%
ack	3	33%	33%	33%	33%	33%	33%	33%	33%	33%
int	67	76%	82%	87%	79%	88%	90%	79%	91%	91%
list	84	75%	94%	94%	80%	96%	96%	81%	99%	99%
suffix	31	94%	100%	100%	94%	100%	100%	97%	100%	100%
reverse	25	72%	88%	88%	84%	100%	100%	84%	100%	100%
permut.	42	48%	71%	71%	60%	79%	81%	62%	86%	86%
sort	42	45%	62%	62%	50%	74%	74%	55%	76%	76%
merges.	24	79%	88%	88%	79%	92%	92%	79%	100%	100%
taut	43	65%	81%	81%	70%	84%	84%	74%	84%	84%

Table 1: Experimental Evaluation

Let us comment these results. The `gcd` Prolog file contains a mutually recursive definition for the predicates `gcd/3` and `gcd_leq/3`. Proving properties of such definitions is currently out of scope of our translation schema.

The `ack` proof file contains the following three properties. The first one is successfully checked. The last two ones cannot be proved with our simplified induction schema. Indeed, the LPTP proofs use an induction inside the top level induction, which is out of scope of our translation schema.

Lemma [*ackermann:types*] $\forall m,n,k\,(\mathbf{S}\,\mathtt{ackermann}(m,n,k) \wedge \mathbf{S}\,\mathtt{nat}(n) \rightarrow \mathbf{S}\,\mathtt{nat}(k))$.

Lemma [*ack:existence*] $\forall m,n\,(\mathbf{S}\,\mathtt{nat}(m) \wedge \mathbf{S}\,\mathtt{nat}(n) \rightarrow \exists k\,\mathbf{S}\,\mathtt{ackermann}(m,n,k))$.

Lemma [*ack:termination*] $\forall m,n,k\,(\mathbf{S}\,\mathtt{nat}(m) \wedge \mathbf{S}\,\mathtt{nat}(n) \rightarrow \mathbf{T}\,\mathtt{ackermann}(m,n,k))$.

5 Related Work

There is quite a few Prolog verification frameworks, see e.g. [7, 10, 2, 24] and more recently [8]. Most of them aim at *paper and pencil* proofs. Although they may offer interesting and elegant methods, the validity of the proofs relies on the usual mathematical writing in natural language, and proofs are not

automatically checked. In our opinion, writing and verifying such hand-written proofs can be a time consuming and error-prone process compared to a push-button approach as the one we present here.

Recently, some quite interesting works have been reported on including datatypes, taking into account the acyclicity of their values, and induction in modern first-oder theorem provers, see, e.g., [4, 12]. We have not yet tested these extensions within our framework.

For Answer Set Programming (a declarative specification language with a Prolog syntax, oriented towards knowledge representation and search problems), [9] describes an approach toward verification in which Vampire checks the equivalence of Answer Set programs.

Some programming languages include automated verification tools *by design*. For example, Dafny [17] makes heavy use of SMT solving. The Why3 system [11] allows to export verification conditions to many automatic and interactive theorem provers.

An earlier account of the integration of automated and interactive theorem proving is described in [1]. As already announced in the introduction, most interactive theorem provers now include the possibility to run some automated theorem provers. Starting with Isabelle, [20, 23, 3, 22], *hammers* can be found in e.g., ACL2, [14], Coq, [6] and Lean, [18].

6 Conclusion

Let us recall the questions of the introduction and propose our answers after this experiment:

- Can we also use the TPTP FOF *Esperanto* to formulate the logic theory Stärk associates to a logic program? Yes. One axiom schema was not implemented: Axiom 3 which forbids rational terms. Another one was partially implemented: Axiom 9 for induction. Actually an inductive argument inside an inductive proof is not possible with our approach. We lose precision but in both cases we stay sound.
- Can we use *off-the-shelf* TPTP provers and obtain automatic proofs in reasonable time? Yes. We use Vampire and the E prover with their most basic options, essentially a timeout. Although Vampire seems to find refutations faster, the E prover can sometimes find proofs while Vampire cannot conclude within the time limit. Hence the two provers are complementary. For the moment, we did not try advanced features offered by the provers like the one proposed in [15].
- Can we get an acceptable success rate with such an approach? Yes. With the E prover and Vampire running in parallel, the average success rate we get from our benchmark is about 83% for a one minute timeout on a standard laptop.

Compared to the efforts one spends while manually, laboriously elaborating certain proofs with an interactive theorem prover, the use of state of the art automated theorem provers is clearly a time-saver. We did not expect such a good success rate for this first experiment. We think there are various reasons that can explain it. Clearly, the computing power of our current laptops is huge and automated theorem provers have been largely improved. Also, thanks to Stärk's ideas, the clean and simple semantics of both the pure subset of Prolog targeted by LPTP and the LPTP specification language – essentially first-order logic – implies a straightforward translation to FOF. Last but not least, Stärk's art of proving, by slicing the proofs of the LPTP library properties into manageable lemmas, certainly has an impact on the success rate we obtain.

Finally, there is room for improvement of the presented work, which can be considered as a first approach towards a hammer for LPTP according to [3]. The first step of a hammer – the *premise selector*, which selects subparts of the LPTP library potentially useful for a proof – and the third step – the *proof*

reconstruction module, which rewrites the proof found by the automatic prover in the LPTP proof format – are yet to be investigated.

Acknowledgements. We thank Manuel Hermenegildo, Daniel Jurjo, Pedro López-Garcia, and Jose Morales for stimulating discussions about LPTP, Geoff Sutcliffe for his help with TPTP and the reviewers for their constructive comments.

References

[1] W. Ahrendt, B. Beckert, R. Hähnle, W. Menzel, W. Reif, G. Schellhorn & P. Schmitt (1998): *Integrating Automated and Interactive Theorem Proving. Automated Deduction — A Basis for Applications: Volume II: Systems and Implementation Techniques*, pp. 97–116. Springer.

[2] K. R. Apt & E. Marchiori (1994): *Reasoning About Prolog Programs: From Modes Through Types to Assertions*. *Formal Aspects Comput.* 6(6A), pp. 743–765, doi:10.1007/BF01213601.

[3] J. C. Blanchette, C. Kaliszyk, L. C. Paulson & J. Urban (2016): *Hammering towards QED*. *J. Formaliz. Reason.* 9(1), pp. 101–148, doi:10.6092/ISSN.1972-5787/4593.

[4] J. C. Blanchette, N. Peltier & S. Robillard (2018): *Superposition with Datatypes and Codatatypes*. In D. Galmiche, S. Schulz & R. Sebastiani, editors: *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, *Lecture Notes in Computer Science* 10900, Springer, pp. 370–387, doi:10.1007/978-3-319-94205-6_25.

[5] K. L. Clark (1977): *Negation as Failure*. In H. Gallaire & J. Minker, editors: *Logic and Data Bases, Symposium on Logic and Data Bases, Centre d'études et de recherches de Toulouse, France, 1977*, Advances in Data Base Theory, Plemum Press, New York, pp. 293–322, doi:10.1007/978-1-4684-3384-5_11.

[6] L. Czajka, B. Ekici & C. Kaliszyk (2018): *Concrete Semantics with Coq and CoqHammer*. In F. Rabe, W. M. Farmer, G. O. Passmore & A. Youssef, editors: *CICM*, *LNCS* 11006, Springer, pp. 53–59, doi:10.1007/978-3-319-96812-4_5.

[7] P. Deransart (1993): *Proof Methods of Declarative Properties of Definite Programs*. *Theor. Comput. Sci.* 118(2), pp. 99–166, doi:10.1016/0304-3975(93)90107-5.

[8] W. Drabent (2016): *Correctness and Completeness of Logic Programs*. *ACM Trans. Comput. Log.* 17(3), p. 18, doi:10.1145/2898434.

[9] J. Fandinno, V. Lifschitz, P. Lühne & T. Schaub (2020): *Verifying Tight Logic Programs with Anthem and Vampire*. *Theory Pract. Log. Program.* 20(5), pp. 735–750, doi:10.1017/S1471068420000344.

[10] G. Ferrand & P. Deransart (1993): *Proof Method of Partial Correctness and Weak Completeness for Normal Logic Programs*. *J. Log. Program.* 17(2/3&4), pp. 265–278, doi:10.1016/0743-1066(93)90033-D.

[11] J.-C. Filliâtre & A. Paskevich (2013): *Why3 - Where Programs Meet Provers*. In M. Felleisen & P. Gardner, editors: *ESOP*, *LNCS* 7792, Springer, pp. 125–128, doi:10.1007/978-3-642-37036-6_8.

[12] M. Hajdú, P. Hozzová, L. Kovács & A. Voronkov (2021): *Induction with Recursive Definitions in Superposition*. In: *Formal Methods in Computer Aided Design, FMCAD 2021, New Haven, CT, USA, October 19-22, 2021*, IEEE, pp. 1–10, doi:10.34727/2021/ISBN.978-3-85448-046-4_34.

[13] ISO/IEC 13211-1 (1995): *Information Technology – Programming Languages – Prolog – Part 1: General Core*.

[14] S. J. C. Joosten, C. Kaliszyk & J. Urban (2014): *Initial Experiments with TPTP-style Automated Theorem Provers on ACL2 Problems*. In F. Verbeek & J. Schmaltz, editors: *International Workshop on ACL2*, *EPTCS* 152, pp. 77–85, doi:10.4204/EPTCS.152.6.

[15] L. Kovács, S. Robillard & A. Voronkov (2017): *Coming to terms with quantified reasoning*. In G. Castagna & A. D. Gordon, editors: *POPL 2017*, ACM, pp. 260–270, doi:10.1145/3009837.3009887.

[16] L. Kovács & A. Voronkov (2013): *First-Order Theorem Proving and Vampire*. In N. Sharygina & H. Veith, editors: *CAV 2013*, *LNCS* 8044, Springer, pp. 1–35, doi:10.1007/978-3-642-39799-8_1.

[17] K. R. M. Leino (2012): *Developing Verified Programs with Dafny*. In R. Joshi, P. Müller & A. Podelski, editors: *VSTTE*, *LNCS* 7152, Springer, p. 82, doi:10.1007/978-3-642-27705-4_7.

[18] P. Lippe (2019): *Lean Hammer*. `https://github.com/phlippe/Lean_hammer`. Accessed: 2025-04.

[19] J. W. Lloyd (1987): *Foundations of Logic Programming, 2nd Edition*. Springer, doi:10.1007/978-3-642-83189-8.

[20] J. Meng & L. C. Paulson (2004): *Experiments on Supporting Interactive Proof Using Resolution*. In D. A. Basin & M. Rusinowitch, editors: *Automated Reasoning - Second International Joint Conference, IJCAR 2004, Cork, Ireland, July 4-8, 2004, Proceedings*, *Lecture Notes in Computer Science* 3097, Springer, pp. 372–384, doi:10.1007/978-3-540-25984-8_28.

[21] F. Mesnard, T. Marianne & É. Payet (2024): *Automated Theorem Proving for Prolog Verification*. In N. S. Bjørner, M. Heule & A. Voronkov, editors: *LPAR 2024 Complementary Volume*, *Kalpa Publications in Computing* 18, EasyChair, pp. 137–151, doi:10.29007/C25R.

[22] L. C. Paulson (2022): *Sledgehammer: some history, some tips*. `https://lawrencecpaulson.github.io/2022/04/13/Sledgehammer.html`. Accessed: 2025-04.

[23] L. C. Paulson & J. C. Blanchette (2010): *Three Years of Experience with Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers*. In G. Sutcliffe, S. Schulz & E. Ternovska, editors: *IWIL*, *EPiC Series in Computing* 2, EasyChair, pp. 1–11, doi:10.29007/36DT.

[24] D. Pedreschi & S. Ruggieri (1999): *Verification of Logic Programs*. *J. Log. Program.* 39(1-3), pp. 125–176, doi:10.1016/S0743-1066(98)10035-3.

[25] S. Schulz, S. Cruanes & P. Vukmirovic (2019): *Faster, Higher, Stronger: E 2.3*. In P. Fontaine, editor: *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, *Lecture Notes in Computer Science* 11716, Springer, pp. 495–507, doi:10.1007/978-3-030-29436-6_29.

[26] R. F. Stärk (1995): *First-order theories for pure Prolog programs with negation*. *Arch. Math. Log.* 34(2), pp. 113–144, doi:10.1007/BF01270391.

[27] R. F. Stärk (1996): *Total Correctness of Logic Programs: A Formal Approach*. In R. Dyckhoff, H. Herre & P. Schroeder-Heister, editors: *ELP'96*, *LNCS* 1050, Springer, pp. 237–254, doi:10.1007/3-540-60983-0_17.

[28] R. F. Stärk (1997): *Formal Verification of Logic Programs: Foundations and Implementation*. In S. I. Adian & A. Nerode, editors: *Logical Foundations of Computer Science, 4th International Symposium, LFCS'97, Yaroslavl, Russia, July 6-12, 1997, Proceedings*, *Lecture Notes in Computer Science* 1234, Springer, pp. 354–368, doi:10.1007/3-540-63045-7_36.

[29] R. F. Stärk (1998): *The Theoretical Foundations of LPTP (A Logic Program Theorem Prover)*. *J. Log. Program.* 36(3), pp. 241–269, doi:10.1016/S0743-1066(97)10013-9.

[30] G. Sutcliffe (2017): *The TPTP Problem Library and Associated Infrastructure - From CNF to TH0, TPTP v6.4.0*. *J. Autom. Reason.* 59(4), pp. 483–502, doi:10.1007/S10817-017-9407-7.

[31] G. Sutcliffe (2023): *The logic languages of the TPTP world*. *Log. J. IGPL* 31(6), pp. 1153–1169, doi:10.1093/JIGPAL/JZAC068.

Investigating the Grounding Bottleneck for a Large-Scale Configuration Problem: Existing Tools and Constraint-Aware Guessing

Veronika Semmelrock
Department of Artificial Intelligence and Cybersecurity
University of Klagenfurt, Austria
veronika.semmelrock@aau.at

Gerhard Friedrich*
Department of Artificial Intelligence and Cybersecurity
University of Klagenfurt, Austria
gerhard.friedrich@aau.at

Answer set programming (ASP) aims to realize the AI vision: The user specifies the problem, and the computer solves it. Indeed, ASP has made this vision true in many application domains. However, will current ASP solving techniques scale up for large configuration problems? As a benchmark for such problems, we investigated the configuration of electronic systems, which may comprise more than 30,000 components. We show the potential and limits of current ASP technology, focusing on methods that address the so-called grounding bottleneck, i.e., the sharp increase of memory demands in the size of the problem instances. To push the limits, we investigated the incremental solving approach, which proved effective in practice. However, even in the incremental approach, memory demands impose significant limits. Based on an analysis of grounding, we developed the method *constraint-aware guessing*, which significantly reduced the memory need.

1 Introduction

Large-scale configurations of electronic systems may comprise more than 200 racks, 1,000 frames, 30,000 modules, 10,000 cables, and 2,000 other units [11]. Electronic systems are typically configured by placing modules into frames and frames into racks. Technical constraints specify which configurations are valid for given user requirements. These requirements may be defined by a set of key modules to be included in a configuration. A configuration specifies the needed components (e.g., racks, frames and modules) and their connections.

Answer set programming (ASP) offers an expressive logical description language that allows a compact problem encoding using a fragment of first-order logic based on all-quantifiers. For a comprehensive definition of ASP's syntax and semantics, we refer to [4]. Efficient solvers make ASP an almost perfect choice for implementing automated configuration systems that enable the user to specify the problem, and the computer solves it [9]. Problem instances of a problem are encoded as an answer-set program, and solutions to a problem instance are extracted from the answer sets (*ASet*s).

Most ASP frameworks follow the so-called ground-and-solve approach. The first-order answer-set program is transformed to a quantifier-free representation (grounding step), and solving operates on this grounded version of the problem description. Since grounding may require super-linear space in the input size (typically defined by a set of facts), applying ASP can become unfeasible due to high memory demands - a phenomenon known as the grounding bottleneck.

Consequently, various methods for mitigating the grounding bottleneck exist, such as lazy grounding with domain-specific heuristics [6], compilation approaches [7], or body-decoupled grounding [3].

*This research was funded by the Austrian Science Fund (FWF) 10.55776/COE12 and by the FFG Austria (www.ffg.at) project SAELING.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 482–495, doi:10.4204/EPTCS.439.33

© V. Semmelrock & G. Friedrich
This work is licensed under the
Creative Commons Attribution License.

Given these proposals, an important question is if ASP is ready for large-scale configuration. In this paper, we report on our evaluation, showing the current status and limitations of the proposed approaches, even on elementary configuration problems of electronic systems.

Interestingly, simple and effective heuristics are available for configuring such systems [12]. These heuristics are exploited for the incremental extension of partial solutions to a complete one, which was the key to applying AI technology for large-scale configuration. Consequently, we investigated the potential of ASP solvers in the convenient case where current ASP solvers can employ such heuristics to extend partial solutions incrementally.

Based on an analysis of the grounded problem instances, we observe that current methods allow choices that are obviously inconsistent and result in significant memory demands. Therefore, we propose the technique of *constraint-aware guessing*, which saves memory by orders of magnitude. Both *constraint-aware guessing* and the compilation approach of [7] substantially increase the efficiency of the incremental generation of solutions.

The paper is structured as follows. We describe a configuration problem comprising the basic elements of electronic systems in Sec. 2. Based on this problem, we analyze current ASP tools and their potential to solve large-scale configurations in Sec. 3. In Sec. 4 we then outline an incremental approach to investigate the limits under the favorable assumption that partial solutions can be incrementally extended. In Sec. 5 we introduce the method *constraint-aware guessing* and show its application to the introduced configuration problem. The results of evaluating the incremental approach are presented in Sec. 6 followed by our conclusions.

2 Configuration of electronic systems

To introduce the problem of configuring electronic systems, Siemens formulated the technology-independent *house configuration problem (HCP)* [13]. In the HCP, the configuration of a house, which includes the entities things (i.e., modules), persons, cabinets (i.e., frames), and rooms (i.e., racks), is required. Existing things and persons, as well as the ownership relation of things belonging to a person is provided as input via `person/1`, `thing/1` and `personTOthing/2` facts, where each person can own any number of things, but each thing belongs to only one person. Domains for the possible rooms and cabinets that can exist are provided as input via `roomDomain/1` and `cabinetDomain/1` facts. The output, i.e., a configuration, is represented by a containment of things in cabinets and cabinets in rooms, as expressed by `cabinetTOthing/2` and `roomTOcabinet/2`, and constrained by specific requirements.

The requirements for a valid configuration are the following: (1) each thing must fit into a cabinet; (2) the maximum capacity of any cabinet is five things; (3) each cabinet must be placed in a room; (4) the maximum capacity of any room is four cabinets; (5) each cabinet belongs to only one person and (6) each room belongs to only one person. Additionally to the standard description of HCP, things and cabinets are numbered, and a thing can only be assigned to a specific cabinet if no higher-numbered cabinet holds a lower-numbered thing. The respective ASP coding is as follows (“:-” is “ ← ”):

```
cabinet(C)   :- cabinetDomain(C), not cabinet_n(C).
cabinet_n(C) :- cabinetDomain(C), not cabinet(C).

room(R)   :- roomDomain(R), not room_n(R).
room_n(R) :- roomDomain(R), not room(R).

cabinetTOthing(C,T)   :- thing(T), cabinetDomain(C), not cabinetTOthing_n(C,T).
cabinetTOthing_n(C,T) :- thing(T), cabinetDomain(C), not cabinetTOthing(C,T).
```

```
:- 1 > #count { C : cabinetTOthing(C,T) }, thing(T). %% Req. 1
:- cabinetTOthing(C1,T), cabinetTOthing(C2,T), C1 < C2. %% Req. 1
:- 6 <= #count { T : cabinetTOthing(C,T), thing(T) }, cabinet(C). %% Req. 2
:- cabinetTOthing(C1,T1), cabinetTOthing(C2,T2), C1 < C2, T1 > T2. %% Additional ordering req.


roomTOcabinet(R,C) :- cabinet(C), roomDomain(R), not roomTOcabinet_n(R,C).
roomTOcabinet_n(R,C) :- cabinet(C), roomDomain(R), not roomTOcabinet(R,C).
:- 1 > #count { R : roomTOcabinet(R,C) }, cabinet(C). %% Req. 3
:- roomTOcabinet(R1,C), roomTOcabinet(R2,C), R1 < R2. %% Req. 3
:- 5 <= #count { C : roomTOcabinet(R,C), cabinetDomain(C) }, room(R). %% Req. 4

personTOcabinet(P,C) :- personTOthing(P,T), cabinetTOthing(C,T). %% Req. 5
:- personTOcabinet(P1, C), personTOcabinet(P2, C), P1 < P2. %% Req. 5
personTOroom(P,R) :- personTOcabinet(P,C), roomTOcabinet(R,C). %% Req. 6
:- personTOroom(P1,R), personTOroom(P2,R), P1 < P2. %% Req. 6

room(R1) :- roomDomain(R1), roomDomain(R2), room(R2), R1 < R2.
cabinet(C1) :- cabinetDomain(C1), cabinetDomain(C2), cabinet(C2), C1 < C2.

room(R) :- roomTOcabinet(R,C).
cabinet(C) :- cabinetTOthing(C,T).
```

As an example, the stated configuration can be created for the following input data:

```
%% input facts
person(1). thing(1). thing(2). personTOthing(1,1). personTOthing(1,2).
person(2). thing(3). thing(4). personTOthing(2,3). personTOthing(2,4).
roomDomain(1). roomDomain(2). cabinetDomain(1). cabinetDomain(2).
%% output facts, representing a configuration
cabinetTOthing(1,1). cabinetTOthing(1,2). roomTOcabinet(1,1).
cabinetTOthing(2,3). cabinetTOthing(2,4). roomTOcabinet(2,2).
```

The presented encoding deliberately avoids the use of choice rules and disjunctive rule heads. This syntactic restriction makes the encoding compatible with various systems that do not support such rules.

3 Benchmarking grounding strategies in ASP: An evaluation of existing methods

We investigated the latest methods for mitigating the grounding bottleneck, such as lazy grounding combined with domain-specific heuristics [6], compilation approaches [7], or body-decoupled grounding [3]. To get the full picture of current approaches, we included the tools *clingo* [14] and *DLV* [2] as representatives for the standard ground-and-solve approach.

Gringo and I-DLV (ground-and-solve): In clingo, the application of the grounder *gringo* is followed by the usage of the solver *clasp*, while the system DLV uses the solver *WASP* after applying *I-DLV* for grounding. Due to I-DLV performing well in some benchmarks, we also evaluated the combination of I-DLV [5] as the groudner with clasp as the solver.

Alpha (lazy grounding with domain-specific heuristics): Alpha [6] is a *lazy-grounding system* that circumvents the grounding bottleneck by interleaving the grounding and solving steps. In this interleaving phase, the ground instances of only those rules are instantiated, which are necessary by the solver's current state. Additionally, Alpha allows declaratively specified domain-specific heuristics to

be included in the encoding to guide the solver and find *ASet*s faster. For the HCP simple and effective heuristics are available and the employed heuristics for our benchmarks can be found at `https://github.com/VeronikaSemmelrock/GroundingBottleneck`.

ProASP (compilation approach): ProASP [7] is a system that partially overcomes the grounding bottleneck by skipping the grounding phase and instead adapting the solving phase via compiling in external propagators subprograms acting as constraints. Custom propagators generated based on the input encoding are injected in the CDCL solver GLUCOSE, which simulates the rules that are not grounded, such as variable initialization. ProASP implements mechanisms to merge the grounding and compilation approaches by allowing the user to specify which parts of the encoding should be compiled or grounded. [7] reports on six different ways to split an encoding by rule type. After testing each possible split with the HCP encoding, using medium-sized instances of 200 things, the constraints split gave the optimal performance w.r.t. runtime and memory consumption. Consequently, in the ProASP experiments, all constraints that do not contain aggregates are compiled, while the remaining rules are grounded.

Newground (body-decoupled grounding): Newground [3] tackles the grounding bottleneck through program rewriting and hybrid-grounding by applying body-decoupled grounding (BDG) to parts of a program. Newground is a system that improves the performance of a traditional ground-and-solve system by applying BDG to rules with many variables, shifting the effort from the grounder to the solver. In BDG, body elements are grounded independently. Because the approach is outperformed by traditional grounding on small rule bodies, newground implements mechanisms for the user to apply BDG on dense, tight rules only while the rest of the encoding is ground, which is called hybrid-grounding. In our experiments, newground is applied exclusively to the constraint `:- cabinetTOthing(C1,T1), cabinetTOthing(C2,T2), C1 < C2, T1 > T2.`, which turned out to be the best hybrid-grounding split. Note that all other rules in the HCP contain fewer variables in the body.

Local search is one of the most successful search strategies if memory consumption of search algorithms is an issue. In [8], the authors describe methods to apply ASP for improving solutions using local search. They assume that an initial solution is provided, e.g., via greedy search. Note that we aim to realize the vision that the user only specifies the problem; thus, this specification should be sufficient to generate an initial solution. Since we seek to find a first solution for the HCP, we refrain from an evaluation of [8].

Experimental setup: To evaluate the performance of state-of-the-art ASP tools under increasing grounding demands, a structured benchmarking setup was developed, focusing on systematically scaling HCP instances. The HCP domain serves as a simplified lower bound for the complexity of typical large-scale configuration problems. Instances were generated synthetically by incrementally adding five persons per step, each with 10 associated things, while proportionally expanding the cabinet and room domains (two cabinets and one room per person) to maintain consistency. This construction was implemented in favor of ASP by adding the exact amount of required cabinets and rooms. One test consisted of searching for the first *ASet* of a problem instance of the HCP with a timeout of one hour. This setup allows for controlled growth in problem size, as each person contributes a constant and known increase in the number of things, while keeping the encoding logic unchanged, enabling analysis of execution time and memory consumption under realistic scaling. Note that things correspond to key modules as described in [11]. These key modules represent the user requirements which are the input for generating configurations.

All experiments were executed on a VM with exclusive access to 32 GB of RAM and four cores of an AMD EPYC 7H12 @ 2.6GHz CPU. The VM was running Ubuntu 22.04.4 LTS x86_64 and had 986 GB of disk space. Tested systems and combinations are Alpha, clingo (gringo and clasp), DLV (I-DLV and WASP), I-DLV+clasp, ProASP and newground. Versions of systems are clingo 5.7.1, DLV 2.1.2,

newground version NaGG (newground2, IJCAI24)[1] and Alpha from [6][2]. The version of ProASP was retrieved from GitHub[3] on 16-12-2024. This is the latest available release at the time of writing.

Analysis: The experimental evaluation (reported in Fig. 1) shows that within the time limit of one hour at

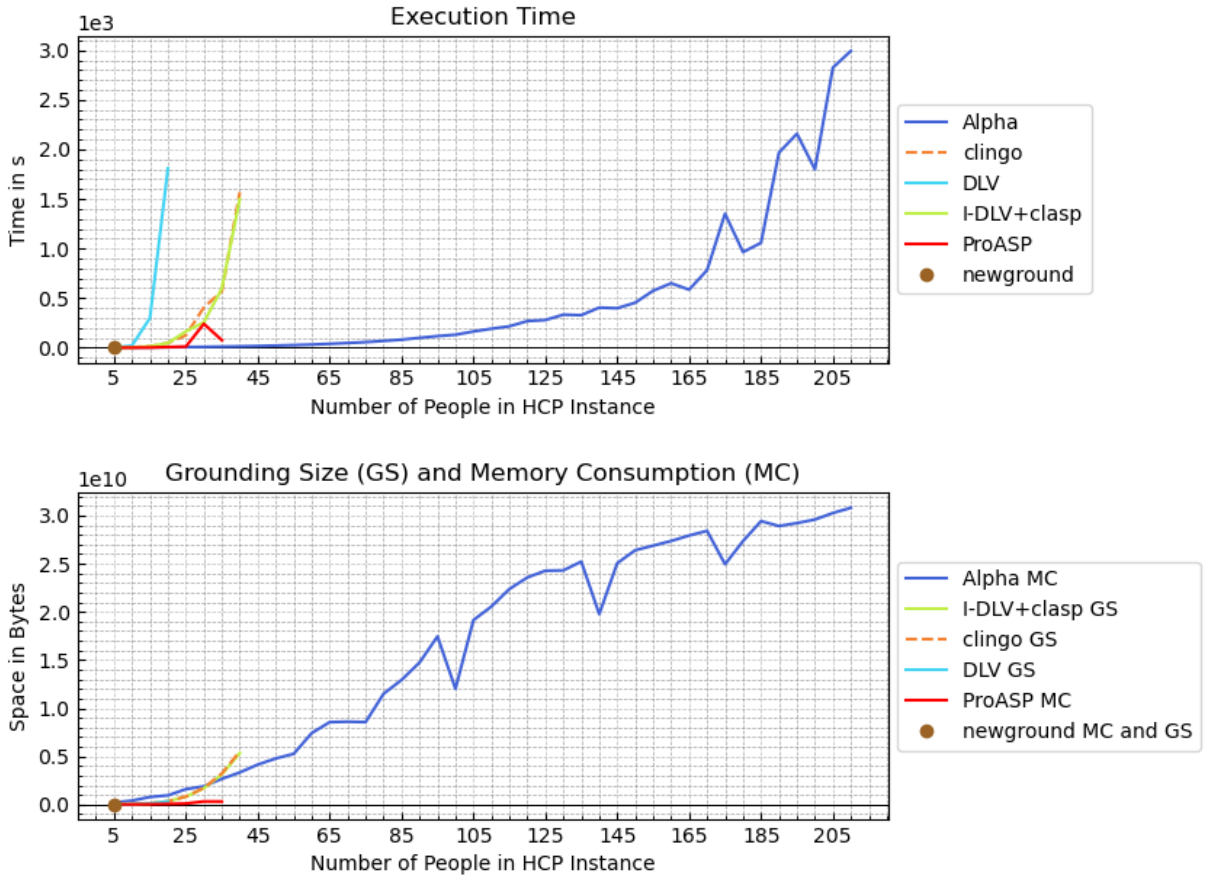


Figure 1: Performance of state-of-the-art systems

most configurations for approx. 2,100 things can be generated by lazy grounding, whereas the standard ground-and-solve approach reaches a maximum of 400 things. DLV, clingo, and I-DLV+clasp show a steep increase in the grounding size. The large size of the grounding led to an error in clasp.

Utilizing its compilation approach, ProASP demonstrates a slightly lower capacity, achieving an instance size of 350 within the same timeframe. However, as the second plot illustrates, the grounding size of ProASP remains comparatively low in relation to the other approaches. Yet, ProASP encounters challenges regarding time efficiency, which hinders its ability to solve larger instances. Based on the information provided by the current version of ProASP, it is currently not possible for the user to determine the reasons for exceeding the time limit.

[1] Available at: `https://github.com/alexl4123/newground/tree/ijcai24-NaGG`

[2] Available at: `https://git-ainf.aau.at/DynaCon/website/tree/master/supplementary_material/2022_JAIR_Domain-Specific_Heuristics`

[3] Available at: `https://github.com/MazzottaG/ProASP`

Newground is only capable of solving the first instance of 50 things and the next instance size of 100 things times out during solving. This limitation can be attributed to the substantial time consumption of the solving step, as the application of newground moves effort from the grounder to the solver.

Note that real-world problem instances may comprise 30,000 things and a much more sophisticated problem description. Although our evaluation favors applying current ASP technology for configuring electronic systems following the module/frame/rack structure, the configuration of large real-world problem instances needs improvements by orders of magnitude.

4 Incremental solving

Our investigation shows that using current technology, a one-step solving approach that passes the complete problem instance to an ASP solver does not scale to the required problem instance size, not even in the simplified HCP encoding. Siemens applies an incremental extension of partial solutions to a full solution [10]. To explore the boundaries, we will again assume a favorable case for ASP tools.

In this favorable scenario, we suppose that a partial solution can be extended to a complete solution and that we make a perfect guess which of the input facts must be added such that no backtracking is required, i.e., no parts of a partial solution must be retracted for generating a complete solution. We assume that a problem is defined by an ASP program P and a set of indexed sets of input facts $I = \{I_1, \ldots, I_n\}$. A solution to the problem is an *ASet* Δ of $P \cup \bigcup_i I_i$.

To generate a solution, we use function INCREMENTALSOLVING (see below). Function $\mathsf{facts}(I)$ transforms an interpretation (rsp. *ASet*) Δ to a set of facts by converting each atom of Δ to a fact.

function INCREMENTALSOLVING(P,$\{I_1, \ldots, I_n\}$)
- ▷ *Inputs: an ASP program P, indexed sets of input facts* $\{I_1, \ldots, I_n\}$ ◁
- ▷ *Output: an answer set* Δ *of* $P \cup \bigcup_i I_i$ ◁
- $\Delta \leftarrow \emptyset$
- $InputFacts \leftarrow \emptyset$
- **for** $i = 1, \ldots, n$ **do**
  - $InputFacts \leftarrow \mathsf{facts}(\Delta) \cup I_i$
  - $\Delta \leftarrow ASPsolver(P \cup InputFacts)$ ▷ *An ASP solver outputs an ASet; input is an ASP program*
- **return** Δ

However, the grounding of an (incrementally extended) ASP program may result in guessing, in which inconsistencies are easy to recognize. Avoiding inconsistent guesses can significantly reduce the size of the grounded program. We illustrate this with a simple example, where modules must be placed in exactly one frame, and frames have a capacity of one module.

```
%% input facts
frame(1). frame(2).
module(1). module(2).
mINf(1,2).
%% modules are placed into frames
%% rule r1
mINf(X,Y) :- not mINf_n(X,Y),
    module(X), frame(Y).
%% rule r2
mINf_n(X,Y) :- not mINf(X,Y),
    module(X), frame(Y).
```

```
%% c1: a frame can hold 1 module
:- module(M1), module(M2), frame(F),
    mINf(M1,F), mINf(M2,F), M1 <> M2.
%% c2: a module can only be placed in 1 frame
:- module(M), frame(F1), frame(F2),
    mINf(M,F1), mINf(M,F2), F1 <> F2.

%% modules must be placed in frames
modulePlaced(X) :- module(X), mINf(X,Y).
:- module(X), not modulePlaced(X).
```

The grounder *gringo* [14] produces the following ground constraints and ground rules for guessing (we omit input facts and duplicate rules for presentation purposes):

```
%%% guessing
mINf(1,1) :- not mINf_n(1,1).
mINf(2,1) :- not mINf_n(2,1).
mINf(2,2) :- not mINf_n(2,2).
mINf_n(1,1 ):- not mINf(1,1).
mINf_n(2,1) :- not mINf(2,1).
mINf_n(2,2) :- not mINf(2,2).
%%% checking
modulePlaced(1).
modulePlaced(2):-mINf(2,1).
modulePlaced(2):-mINf(2,2).
:- not modulePlaced(2).
:- mINf(1,1).
:- mINf(2,1),mINf(2,2).
:- mINf(2,2),mINf(2,1).
:- mINf(1,1),mINf(2,1).
:- mINf(2,1),mINf(1,1).
:- mINf(2,2).
```

Note, that `module(1)` is already placed in `frame(2)`. Asserting `mINf(1,1)` or `mINf(2,2)` by guesses will lead to violated constraints. If we avoid these guesses that will not lead to a consistent solution, we can remove grounded constraints and rules.

5 Constraint-Aware Guessing (CAG)

In this section, we introduce the method for *constraint-aware guessing* to filter guesses to reduce the size of grounded programs. Note that problem instances are encoded by a program P and every *ASet* corresponds to a solution. We assume a partitioning of $P = P^{check} \cup P^{guess}$ s.t. for every interpretation I of P which does not correspond to a solution, $P^{check} \cup \mathsf{facts}(I)$ is unsatisfiable.

Step 1: Let rule r be a rule of P^{guess} with a nonempty head whose body contains default negated atoms, i.e., $\mathsf{body}^-(r) \neq \emptyset$. Consequently, this rule is subject to a guess. Let c be a constraint of P where the atom a_r in $\mathsf{head}(r)$ unifies with an atom a_c in $\mathsf{body}^+(c)$ s.t. $a_r = a_c\gamma$ where γ is a substitution that maps terms to terms (e.g., variables to variables). We assume w.l.o.g. that if atoms a_r, a_c are atoms with arguments, these arguments are variables. Arguments that are not variables (e.g., constants) can be emulated by adding equality atoms to the body. In addition, we assume ground predicates bind the variables in $\mathsf{body}^+(c)$ to assure safety, and the names of variables in r and c are different.

For each atom $a_c \in \mathsf{body}^+(c)$ where $a_r = a_c\gamma$ we generate a *filter condition* $b(r, a_c) = (\mathsf{body(c)} - a_c)\gamma$. For constraint `c1` and rule `r1` in our example this results in

b_1 : { `module(X), module(M2), frame(Y), mINf(M2,Y), X <> M2` }
b_1' : { `module(M1), module(X), frame(Y), mINf(M1,Y), M1 <> X` }

Note b_1 and b_1' are equivalent, aside from renaming variables `M1, M2`. Therefore, we consider only b_1 subsequently. For constraint `c2` in our example we generate

b_2 : { `module(X), frame(Y), frame(F2), mINf(X,F2), Y <> F2` }

As for constraint `c1`, we do not consider the equivalent filter condition.

Step 2: Observe that if condition $b_1\gamma$ and $\mathsf{body}(\mathtt{r1})\gamma$ are true in an interpretation I for a substitution γ then constraint `c1` is unsatisfiable.

Provided P contains a program P^{check}, we can add the negation of all filter conditions to the body of r, which does not reduce the set of *ASet*s of P but reduces the ground instantiations of r. Let $B(r,P)$ be all the filter conditions for a rule r for all the constraints $c \in P$. We replace r by its filtered version r^f defined as $\mathsf{head}(r) :\text{-} \mathsf{body}(r) \bigcup_{b(r,a_c) \in B(r,P)} \{\text{not } b(r,a_c)\}$. Since the default negation is not applicable for a set of atoms, we employ the expression `#count {1:`$b(r,a_c)$`}` < 1 for not $b(r,a_c)$ if the filter condition $b(r,a_c)$ does not contain aggregates. Note that syntactically it is not possible for aggregates to contain

aggregates. If $b(r,a_c)$ is a single aggregate atom, we can negate the aggregate function, e.g., replacing $<$ by $\geq$. If $b(r,a_c)$ comprises multiple atoms where at least one atom is an aggregate atom, we could introduce auxiliary rules and atoms to move an aggregate outside an aggregate. However, this results in an increase of the size of the grounded program.

Proposition: Let r be a rule with $\mathtt{body}^-(r) \neq \emptyset$, r^f its filtered version, $P^{check} = P \setminus P^{guess}$ a program as described, and program $P^f = (P \setminus \{r\}) \cup \{r^f\}$ (i.e., rule r is exchanged by r^f). Every *ASet* (solution) of P is an *ASet* (solution) of P^f. By the properties of P^{check} all *ASet*s of P^f correspond to a solution.

Proof: For every *ASet* of P where $\mathtt{body}(r)\sigma$ is true for some σ, $(b(r,a_c)\sigma)\delta$ must be false for all possible ground substitutions δ. Otherwise a constraint would be unsatisfied. Consequently, we can extend r to $r^f : \mathtt{head}(r) \mathtt{:-} \mathtt{body}(r) \cup \{not\ b(r,a_c)\}$. and $\mathtt{body}(r^f)$ is true in *ASet*. Therefore, if I is an *ASet* for P then I is an *ASet* for P^f. However, P^f may comprise *ASet*s which are not *ASet*s of P. Yet, P^{check} assures that these *ASet*s correspond to solutions. □

In our example, we extend rule `r1` with constraint-aware filter conditions as follows:

```
mINf(X,Y) :- not mINf_n(X,Y), module(X), frame(Y),
 #count{1: module(X), module(M2), frame(Y), mINf(M2,Y), X <> M2 } < 1,
 #count{1: module(X), frame(Y), frame(F2), mINf(X,F2), Y <> F2 } < 1.
```

This results in a significantly smaller grounding generated by *gringo* for the guessing and checking part of the grounded program (input facts are omitted):

```
%%% guessing
mINf(2,1):- not mINf_n(2,1),
    #delayed(1),#delayed(2).
mINf_n(1,1).
mINf_n(2,1):- not mINf(2,1).
mINf_n(2,2).
#delayed(1) <=> #count{}
#delayed(2) <=> #count{}
%%% checking
:- not modulePlaced(2).
modulePlaced(1).
modulePlaced(2):- mINf(2,1).
```

The delay atoms are added by *gringo* as auxiliary constructs for dealing with aggregates.

Note that it is possible to generate new constraints by replacing atoms a_c in a constraint with bodies of rules whose head a_r matches a_c, allowing additional filter conditions to be generated. For the HCP we consider the follwoing generated constraint

```
:- personTOthing(P1,T1), cabinetTOthing(C,T1),
   personTOthing(P2,T2), cabinetTOthing(C,T2), P1 < P2.
```

Applying constraint-aware guessing to HCP: To demonstrate and evaluate the proposed method, which constitutes a general strategy for rewriting ASP encodings, it was applied to the HCP as a representative case study. The complete resulting encoding of adding filter conditions to HCP rules can be found in `https://github.com/VeronikaSemmelrock/GroundingBottleneck`. As an example, the rule for guessing `cabinetTOthing/2` is extended as follows:

```
cabinetTOthing(C,T) :- thing(T), cabinetDomain(C), not cabinetTOthing_n(C,T),
    #count{1: cabinetTOthing(C1,T), C < C1 } <1, %% Req. 1
    #count{1: cabinetTOthing(C1,T),  C1 < C } <1, %% Req. 1
    #count{1: cabinetTOthing(C1,T1), C1 < C, T1 > T } <1, %% Additional ordering requirement
    #count{1: cabinetTOthing(C1,T1), C < C1, T > T1 } <1, %% Additional ordering requirement
    #count{1: personTOthing(P1,T), personTOthing(P2,T2), cabinetTOthing(C,T2), P1<P2}<1, %Req. 5
    #count{1: personTOthing(P1,T1), personTOthing(P2,T), cabinetTOthing(C,T1), P1<P2}<1. %Req. 5
```

This rule uses multiple aggregates derived from constraints of mentioned requirements, enforcing consistency in assignments and ordering between things, cabinets, and persons. Note that both derived `#count`-aggregates of requirement 1 combined behave like `#count{1: cabinetTOthing(C1,T), C`

`!= C1} < 1`, which ensures that each thing T is placed into at most one cabinet. The two `#count`-aggregates of requirement 5 are created via substitution and combination of the two rules of requirement 5 of the unchanged HCP encoding (see Section 2).

6 Benchmarking incremental solving

The implementation of the function INCREMENTALSOLVING (see Section 4) employs an ASP solver. Each invocation of the ASP solver is stateless; that is, no information is retained between consecutive calls and the solver is reinitialized from scratch for every run. In our evaluations, we applied clingo, DLV, newground, and ProASP as well as the combination of I-DLV and clasp, resulting in *incremental-clingo*, *incremental-DLV*, *incremental-newground*, *incremental-ProASP*, and *incremental-I-DLV+clasp*. The implementation of CAG for HCP builds on the incremental solving system incremental-clingo. In contrast to incremental-clingo, which operates on the HCP encoding of Section 2, CAG employs the adapted encoding including filter conditions, detailed in Section 5.

An attempt was made to combine the CAG approach with incremental-ProASP; however, the integration failed due to ProASP's syntactic limitations, particularly its handling of aggregates. These issues prevented correct parsing and compilation, making the combination infeasible in its current form.

Experimental setup: The experimental setup for incremental solving differs from that in Section 3 solely regarding the tested systems. The function INCREMENTALSOLVING was implemented via a Python script. Initially, the script generates all the facts for a problem instance (as in Section 3), but subsequently divides them into fact batches. These fact batches then serve as the input facts $I = \{I_1, \ldots, I_n\}$, which are added in each iteration of the incremental solving process to extend the solution. In the HCP, it is possible to configure all the things of one person independently from all other people's facts; thus, a fact batch can contain facts of one person up to several people, i.e., the `person/1`, `thing/1`, and `personTOthing/2` facts as well as the extensions of the domains via `roomDomain/1` and `cabinetDomain/1` facts. The Python script implements code to call the solvers, providing the fact batch and the problem encoding, and extracting the found *ASet*. Next, this *ASet* is parsed and added as facts to the fact batch of the next iteration. The decomposition of the problem instance into fact batches introduces a variable `persons-per-iteration (PPI)`, which describes the number of people whose facts are added to one fact batch for one iteration. All tools were initially tested using a `PPI` of five, as this showed the best performance in CAG. The optimal `PPI` was determined for instances up to 100 people. In this setting, a `PPI` of five is also optimal for incremental-ProASP. Employing ProASP for incremental solving, the constraints split was again identified as the most efficient and least memory-consuming approach. Furthermore, incremental-clingo, which is after CAG the most effective ground-and-solve system, was additionally optimized by determining an optimal `PPI`. The optimized `PPI` for incremental-clingo was determined to be one which then was employed in a separate benchmark test, following the same principles. We denote these results in our graphs by `PPI-opt. incremental-clingo`.

Comparative analysis: The plots in Figure 2 demonstrate the results of the state-of-the-art tools and systems and all incremental solving tools, including CAG. The results show that CAG solves the largest instances, followed by incremental-ProASP.

There was a substantial improvement in the incremental application of clingo, DLV, I-DLV+clasp, newground, and ProASP compared to their one-step application. The most significant relative enhancement was observed in ProASP, where the incremental application of ProASP on the HCP successfully solved a 1.585 % larger instance within one hour. Due to the limited transparency of the ProASP's internal mechanisms, it is challenging to precisely attribute the source of performance improvements

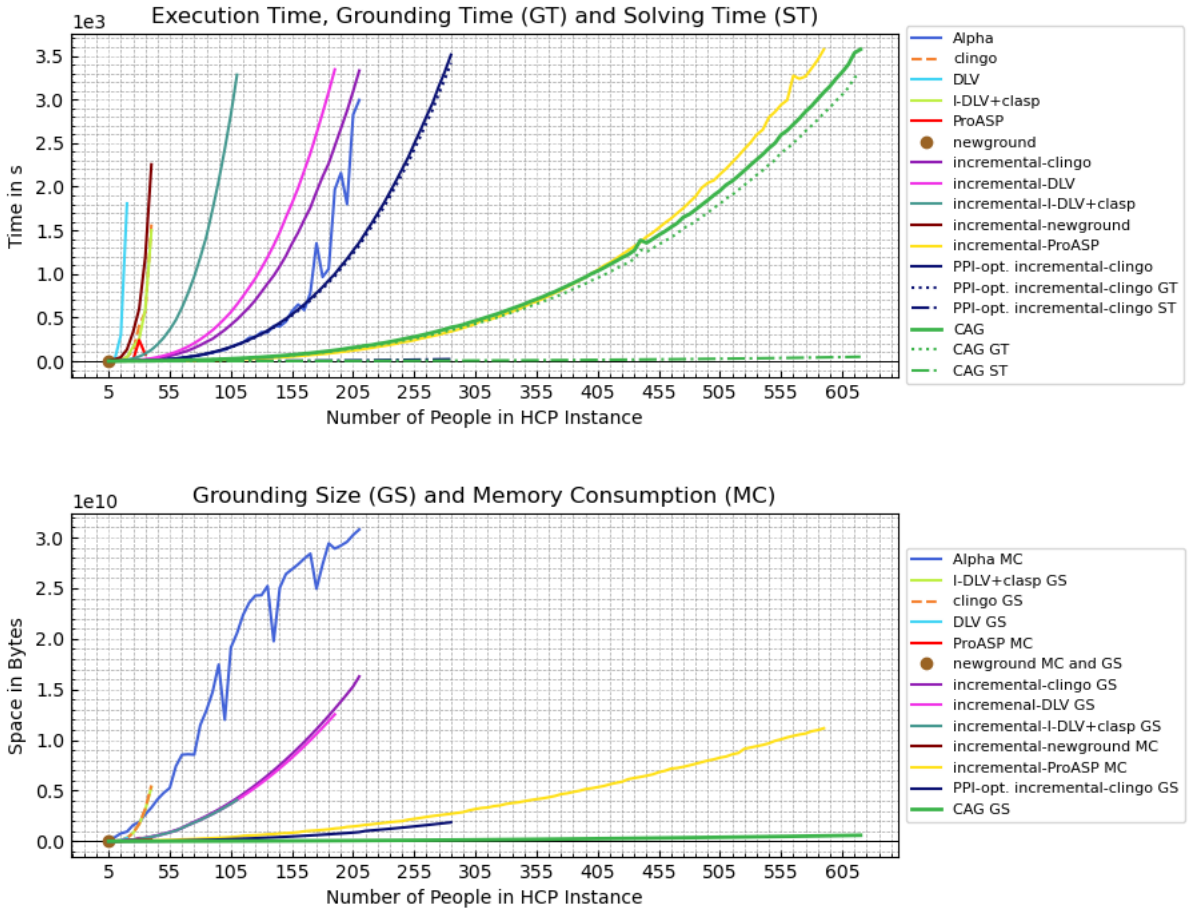


Figure 2: Benchmarks of all tools and approaches

observed when it is applied incrementally. The second largest relative improvement was observed with DLV. Newground also exhibited a substantial improvement by 700 % being able to solve a maximum of 400 things. In comparison, the incremental application of clingo on the HCP resulted in a 425 % improvement in instance sizes solved, while incremental-I-DLV+clasp reached 175 % larger instances.

Furthermore, incremental-clingo is depicted in the plot once with a PPI of 5, like all other approaches, and once with its optimized PPI value of 1. The optimization of the PPI value for incremental-clingo resulted in the approach being able to solve an instance with 2,850 things instead of 2,100. Note that the second plot in Figure 2 depicts the grounding size in the final iteration, which is the largest during the entire execution. In the case of incremental-clingo, this illustrates the effort involved in adding the facts of five individuals, each introducing 10 things. Whereas the PPI-opt. incremental-clingo values show the effort of adding the facts of only one person. Thus, the lines of incremental-clingo are not comparable.

A comparison between the two best performing ground-and-solve approaches, PPI-opt. incremental-clingo and CAG, shows a performance advantage of 118 % for CAG, where PPI-opt. incremental-clingo can solve an instance size of 2,850 things in one hour, while CAG can solve an instance size of 6,200 things. It is possible to compare the grounding size of incremental-clingo and CAG, as both have a PPI of 5. This shows a 99.6 % reduction in the grounding size of CAG in the last iteration of the largest common instance of 2,100 things. A naive comparison of PPI-opt. incremental-clingo and CAG shows that the grounding size of CAG is 93.1 % smaller when comparing the largest common instance although CAG grounds five times as many new facts in the last iteration as PPI-opt. incremental-clingo.

A comparison of the two approaches that demonstrated the highest overall performance, CAG and incremental-ProASP, revealed that their performance was comparable despite the difference in their underlying mechanisms. Incremental-ProASP reached an instance size of 5,900, while CAG successfully solved 6,200 things within one hour. As both have an optimal PPI of 5, comparing their performance regarding grounding size for CAG and memory consumption for incremental-ProASP is possible. In their largest common instance of 5,900 things, CAG has a grounding size that is 95 % smaller than the memory consumption of incremental-ProASP. This clearly demonstrates the advantage of CAG regarding memory demands when instance sizes are increased.

Moreover, the first plot in Fig. 2 includes separate metrics of total grounding time (GT) and solving time (ST) for the CAG approach, which proved to be the best approach overall, and the PPI-opt. incremental-clingo approach, which was the second best ground-and-solve approach. CAG solves the largest instances and improves all performance metrics by lowering execution, grounding, and solving time. In the CAG approach, approximately 94 % of the execution time is allocated to grounding, while solving accounts for 1.5 % of the total execution time across all instance sizes. The remaining 4.5 % of the execution time is attributed to the Python script implementing the incremental solving mechanism. A similar trend is observed in the case of PPI-optimized incremental-clingo, with grounding accounting for approximately 97 % of the total execution time. The remaining 3 % is divided into approx. 0.6 % solving time and approx. 2 % time for executing the Python script.

Other methods for incremental solving and problem decompositions: In [15] multi-shot ASP solving was introduced to deal with continuously changing programs. Multi-shot ASP solving aims to avoid redundancies in relaunching grounder programs and benefits from the solver's learning capacities. In our incremental solving process, we extend program $P_k = P \cup \bigcup_{i=1\ldots k} I_i$ by input facts I_{k+1} and an *ASet* Δ_k for P_k. Consequently, all grounded rules of P_k are deleted or reduced to facts, and a new problem is solved. Therefore, the benefits of reusing the grounding of P_k by multi-shot ASP are limited. Adding Δ_k to P_k by facts is impossible because this extension is not compositional, i.e., an atom would be defined in two different modules. Adding Δ_k as externals has no effect because the atoms in Δ_k are output atoms

in P_k. It is possible to add Δ_k to P_k via constraints, however, this entails a substantial reformulation of the problem encoding, which is preferably done automatically. This reformulation is beyond the scope of this investigation and is part of future work.

In [1] an approach (D-FLAT) based on a tree-decomposition and dynamic programming employing ASP was introduced. Dynamic programming exploits a tree structure of the problem. It assumes that (partial) solutions for subsets of nodes in a graph can be combined so that a global solution can finally be generated by combining the partial solutions. Consequently, for a given problem, the design of the appropriate data structures and answer-set programs employed in dynamic programming is required. The transformation from ASP to SAT (assuming appropriate syntactical restrictions of ASP) can realize the general applicability of dynamic programming for ASP because D-FLAT can solve SAT problems. However, such an approach does not apply to HCP for large instances because of the grounding size.

7 Conclusion and discussion

We investigated the potential of current ASP methods for configuring large-scale electronic systems as described in [11], which may comprise more than 30,000 modules, with a simplified HCP encoding serving as a lower bound for the complexities of industry problems, aiming to create a favourable case for ASP systems. For our evaluation, we defined a configuration problem that follows the standard module/frame/rack structure of such systems [13]. Still, we did not consider cables, different module types, or complex technical constraints (e.g., balancing physical quantities). Within a time limit of one hour, the most efficient ground-and-solve approach manages to configure systems in the size of approximately 400 modules (i.e., things in HCP). Due to the superlinear increase in space demands and runtime as the input size grows, polynomial regression predicts a grounding size of more than 5 exabytes and runtime of more than 10,000 years for 30,000 modules for the ground-and-solve approaches. A special case is lazy-grounding, which can push the limit to approx. 2,000 modules; however, effective heuristics must be available. We, therefore, investigated the case, which seems to be shared in many configuration problems [12, 10], where configuration problem instances can be incrementally solved. Incremental solving pushes the limit to approximately 3,000 modules for incremental-clingo and can be further improved by *constraint-aware guessing*, which manages to double the size for the one hour time limit and has a significantly lower increase of memory demands (approx. 99% savings). The incremental application of ProASP turned out to be comparable to CAG regarding the size of the solvable instances within a one hour time limit. However, memory demands of CAG are significantly lower compared to incremental-ProASP (approx. 95% savings).

Note that we assumed a configuration problem with few technical constraints and a low variety of component types. The challenge of solving such problems is the size of the problem instances and their solutions. Performing guesses that allow a solution is easy in many practical configuration problems [10]. Whether current solving approaches will scale up for such domains or radical new approaches are required remains an essential open question.

Future work includes a systematic investigation of grounding behavior in multi-shot ASP systems, similar to the analysis conducted in this work. Additionally, exploring how CSP solvers handle grounding for problems such as the HCP, which represents a lower bound in terms of complexity, may provide further insights into solver behavior and efficiency across paradigms.

References

[1] Michael Abseher, Bernhard Bliem, Günther Charwat, Frederico Dusberger, Markus Hecher & Stefan Woltran (2014): *The D-FLAT System for Dynamic Programming on Tree Decompositions*. In Eduardo Fermé & João Leite, editors: *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*, *Lecture Notes in Computer Science* 8761, Springer, pp. 558–572, doi:10.1007/978-3-319-11558-0_39.

[2] Weronika T Adrian, Mario Alviano, Francesco Calimeri, Bernardo Cuteri, Carmine Dodaro, Wolfgang Faber, Davide Fuscà, Nicola Leone, Marco Manna, Simona Perri et al. (2018): *The ASP system DLV: advancements and applications*. *KI-Künstliche Intelligenz* 32, pp. 177–179, doi:10.1007/S13218-018-0533-0.

[3] Alexander Beiser, Markus Hecher, Kaan Unalan & Stefan Woltran (2024): *Bypassing the ASP bottleneck: hybrid grounding by splitting and rewriting*. In: *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 3250–3258. Available at `https://www.ijcai.org/proceedings/2024/360`.

[4] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca & Torsten Schaub (2020): *ASP-Core-2 Input Language Format*. *Theory Pract. Log. Program.* 20(2), pp. 294–309, doi:10.1017/S1471068419000450.

[5] Francesco Calimeri, Davide Fuscà, Simona Perri & Jessica Zangari (2017): *I-DLV: The new intelligent grounder of DLV*. *Intelligenza Artificiale* 11(1), pp. 5–20, doi:10.3233/IA-170104. Available at `https://journals.sagepub.com/doi/abs/10.3233/IA-170104`.

[6] Richard Comploi-Taupe, Gerhard Friedrich, Konstantin Schekotihin & Antonius Weinzierl (2023): *Domain-specific heuristics in answer set programming: A declarative non-monotonic approach*. *Journal of Artificial Intelligence Research* 76, pp. 59–114, doi:10.1613/JAIR.1.14091.

[7] Carmine Dodaro, Giuseppe Mazzotta & Francesco Ricca (2024): *Blending grounding and compilation for efficient ASP solving*. In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 21, pp. 317–328, doi:10.24963/KR.2024/30.

[8] Thomas Eiter, Tobias Geibinger, Nelson Higuera Ruiz, Nysret Musliu, Johannes Oetsch, Dave Pfliegler & Daria Stepanova (2024): *Adaptive large-neighbourhood search for optimisation in answer-set programming*. *Artificial Intelligence* 337, p. 104230, doi:10.1016/J.ARTINT.2024.104230.

[9] Andreas Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe & Erich C Teppan (2018): *Industrial applications of answer set programming*. *KI-Künstliche Intelligenz* 32(2), pp. 165–176, doi:10.1007/S13218-018-0548-6.

[10] Andreas A. Falkner, Gerhard Friedrich, Alois Haselböck, Gottfried Schenner & Herwig Schreiner (2016): *Twenty-Five Years of Successful Application of Constraint Technologies at Siemens*. *AI Mag.* 37(4), doi:10.1609/AIMAG.V37I4.2688.

[11] G. Fleischanderl, G.E. Friedrich, A. Haselbock, H. Schreiner & M. Stumptner (1998): *Configuring large systems using generative constraint satisfaction*. *IEEE Intelligent Systems and their Applications* 13(4), pp. 59–68, doi:10.1109/5254.708434.

[12] Gerhard Fleischanderl & A Haselböck (1996): *Thoughts on partitioning large-scale configuration problems*. In: *AAAI 1996 Fall Symposium Series*, pp. 1–10.

[13] Gerhard Friedrich, Anna Ryabokon, Andreas A. Falkner, Alois Haselböck, Gottfried Schenner & Herwig Schreiner (2011): *(Re)configuration based on model generation*. In Conrad Drescher, Inês Lynce & Ralf Treinen, editors: *Proceedings Second Workshop on Logics for Component Configuration, LoCoCo 2011, Perugia, Italy, 12th September 2011*, *EPTCS* 65, pp. 26–35, doi:10.4204/EPTCS.65.3.

[14] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Marius Lindauer, Max Ostrowski, Javier Romero, Torsten Schaub, Sven Thiele & Philipp Wanko (2019): *Potassco guide version 2.2.0*. Available at `https://github.com/potassco/guide/releases/tag/v2.2.0`. Retrieved from `https://github.com/potassco/guide/releases/tag/v2.2.0`.

[15] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2019): *Multi-shot ASP solving with clingo*. *Theory Pract. Log. Program.* 19(1), pp. 27–82, doi:10.1017/S1471068418000054.

On Zeno-like Behaviors in the Event Calculus with Goal-directed Answer Set Programming*

Ondřej Vašíček[1] Joaquin Arias[2] Jan Fiedor[1,6] Gopal Gupta[3]
Brendan Hall[4] Bohuslav Křena[1] Brian Larson[5] Tomáš Vojnar[1,7]

[1]Brno University of Technology, CZ [2]CETINIA, Universidad Rey Juan Carlos, Spain [3]UT Dallas, USA
[4]Ardent Innovation Labs, USA [5]Multitude Corp., USA [6]Honeywell International s.r.o., CZ [7]Masaryk University, CZ

It has been argued that Event Calculus (EC) is suitable for modeling high-level specifications of safety-critical cyber-physical systems. The primary advantage lies in the rather small semantic gap between EC models and requirements expressed in a semi-formal natural language. Moreover, its use of continuous time and variables avoids imprecision that stems from discretization. In the past, we have shown that a goal-directed ASP system can be used for implementing these EC models. However, precise representation of time as an infinitesimally divisible continuous quantity leads to Zeno-like behaviors and to non-termination in such a system. In this work, we model a number of well-known example problems from the literature to systematically study various natural EC modeling patterns that yield these Zeno-like behaviors, and propose ways to deal with them. Moreover, we also propose a technique to automatically detect all such cases.

1 Introduction and Motivation

Requirements engineering is a wide field of research due to its various applications in critical sectors, such as medical or aerospace, where it is necessary to verify the correct operation of cyber-physical systems (CPS). Their development starts with the specification of *system requirements*, which is by many considered the most crucial part of the development process. The importance is clearly illustrated by the results of the AVSI SAVI project [19] which show that 70% of errors are introduced during the system specification and design phases, yet most of them are detected later, during the testing of system's concrete implementation. The main reason the errors are not detected earlier is that the system specification describes a whole set of feasible solutions from which one concrete solution is eventually chosen to be implemented. Such a single solution is much easier to verify compared to a potentially huge set of solutions to validate in the early phases, mainly because we can make assumptions, e.g., that the CPS operates in discrete time. When reasoning about all feasible solutions, we need to represent their real-world behavior as closely as possible which includes modeling the system behavior in continuous time and space. However, this introduces issues such as Zeno behavior where, theoretically, an infinite number of events can occur in a finite interval—an unrealistic behavior which makes modeling difficult.

As argued in [18], a promising way to reason about system requirements is to transform them into *Event Calculus* (EC) [12] and use its semantics for the verification and validation. EC is particularly suitable since its semantics follows how a human would think of the requirements, which makes the semantic gap between EC and the requirements near-zero. This allows EC to faithfully reason about the behavior defined by the requirements without being tainted by design or implementation decisions, such

*The work was supported by the project 23-06506S of the Czech Science Foundation and the FIT BUT internal project FIT-S-23-8151; grant VAE (TED2021-131295B-C33) funded by MCIN/AEI/10.13039/501100011033 and by the "European Union NextGenerationEU/PRTR", grant COSASS (PID2021-123673OB-C32) funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe"; by US NSF Grants IIS 1910131 and grants from industry through the UT Dallas Center for Applied AI and Machine Learning.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 496–510, doi:10.4204/EPTCS.439.34

© O. Vašíček et al.
This work is licensed under the
Creative Commons Attribution License.

as “designing" states, transitions, or decomposition into sub-systems as is the case, e.g., with automata-based approaches. The capabilities of this approach have already been shown on a train gate controller system [17] and in our prior work [18], where we managed to discover a number of inconsistencies and a violation of a safety property in the specification of a safety-critical medical device. Both of the above works use *s(CASP)*—a goal-directed, grounding-free reasoner for Answer Set Programming (ASP) with constraints [4]. The grounding-free nature and support for constraint solving allow s(CASP) to reason about *continuous time* and *continuous quantities*, which are both crucial for accurately and realistically representing CPS behavior. The goal-directed nature further allows s(CASP) to provide *explainable results*, which is crucial when using results of automated reasoning as evidence for certification, e.g., when building assurance cases [13]. However, similar to Prolog, s(CASP) can suffer from *non-termination*, especially when reasoning in continuous time, which both of the above works struggled with.

In this work, we identify and systematically classify a number of general *Zeno-like behaviors* which cause non-termination in goal-directed reasoning under continuous time. We have implemented all relevant examples from the literature, including fairly complex ones like the water tanks [2] and real safety-critical systems [18]. We show how to address the identified classes of Zeno-like behaviors so that the non-termination issues are avoided. More precisely, by *Zeno-like* behavior, we mean behavior that makes the reasoning *explore* an infinite number of events with *infinitesimal* time intervals in between, but an infinite number of events do not *actually occur*, making it only similar to Zeno behavior. It is caused by common EC modeling patterns, e.g., preventing repeated triggering, some of which are specific to goal-directed reasoning, e.g., terminating specific trajectories. We have identified a unifying feature of all these behaviors, a *Zeno-descending chain of events* (Section 3), and propose a technique for detecting it during reasoning (Section 3.2). We introduce each of the general Zeno-like behaviors using examples from the literature or their variations (Section 4 and 5) and we propose solutions for each of them. Then, we discuss the water tanks example which combines all the behaviors as sub-problems and, in addition, exhibits real Zeno behavior (Section 6). This example illustrates all the prior solutions combined and solving the Zeno-like sub-problems leads to the real Zeno behavior being dealt with as well.

2 Background & Related Work

Zeno’s paradoxes of motion were formulated by the ancient philosopher Zeno of Elea: his Racetrack paradox (or dichotomy of motion) says that a runner will never reach the finish line because they must run half of the distance first, then half of what is remaining, etc. Zeno executions or Zeno runs are a well-known problem in timed/hybrid systems and automata. A common definition of Zeno behavior is that an infinite number of events or state transitions occur in a finite time interval. Zeno behaviors do not occur in real physical systems and are a byproduct of abstractions. According to [20], many works simply impose non-Zeno assumptions, such as [7]. A way to deal with Zeno behaviors is to remove them by modifying the system using techniques such as regularization [9]. Further research proposes techniques for detecting Zeno behavior [6, 14], or sufficient conditions for the presence (or lack of) Zeno behavior [20, 10, 8]. However, as far as we know, a systematic exploration of Zeno behaviors in EC has not been conducted yet. For example, in Section 4, we show that the bank account example [12] fundamentally has no solution in continuous time leading to Zeno-like behavior in goal-directed reasoning.

2.1 s(CASP) and Event Calculus

The s(CASP) system [4] extends the expressiveness of ASP systems, based on the stable model semantics [5], by including predicates, constraints among non-ground variables, uninterpreted functions, and, most importantly, a top-down, query-driven execution strategy. This makes it possible to return answers with non-ground variables and to compute partial models by returning only the necessary fragment of

```
1 fluent(light_on).
2 event(turn_light_on).   event(turn_light_off).
3 initiates(turn_light_on,  light_on, T).
4 terminates(turn_light_off, light_on, T).
5 % automatically created can_* rules
6 can_initiates(turn_light_on,  light_on, T).
7 can_terminates(turn_light_off, light_on, T).
```

(a) Domain model (ex1/model.pl).

```
1 initiallyN(light_on).
2 happens(turn_light_on,  10).  happens(turn_light_off, 20).
3 ?- holdsAt(light_on,    T).  % expected: T #> 10,T #=< 20
```

(b) Narrative and a query with the expected answer.

```
1  holdsAt(Fluent, T2) :-
2    T1 #> 0,   T1 #< T2,
3    can_initiates(Event, Fluent, T1),
4    happens(Event, T1),
5    initiates(Event, Fluent, T1),
6    not_stoppedIn(T1, Fluent, T2).
7
8  stoppedIn(T1, Fluent, T2) :-
9    T1 #< T,   T #< T2,
10   can_terminates(Event, Fluent),
11   happens(Event, T),
12   terminates(Event, Fluent, T).
13 not_stoppedIn(T1, Fluent, T) :- ...
```

(c) EC axioms at axioms/bec_scasp-small.pl.

Figure 1: s(CASP) encoding of Example 1 (light on/off).

a stable model to support the answer, which can include the full proof tree making them fully explainable. In s(CASP), thanks to its constructive negation, `not p(X)` can return bindings for `X` for which the call `p(X)` would have failed. Thanks to the interface of s(CASP) with constraint solvers, sound non-monotonic reasoning with constraints is possible. Like other ASP implementations and unlike Prolog, s(CASP) handles non-stratified negation and returns the corresponding (partial) stable models, e.g., for `p :- not q. q :- not p.`, under the stable model semantics there are two possible models for this *even loop*, with either `p` or `q` being true.

The Event Calculus (EC) is a formalism for reasoning about events and change [16, 12] of which there are several versions. We use the Basic Event Calculus (BEC) [15, 11]. There are three basic concepts in EC: (i) an *event* is an action that may occur in the world, (ii) a *fluent* is a time-varying property of the world, (iii) a *timepoint* is an instant of time. Events may happen at a timepoint. Fluents have a truth value at any timepoint and may have quantities associated with them. They change discretely via events or continuously via *trajectories*. An EC formulation of an example consists of a universal theory, a domain model, and a narrative. The theory is a conjunction of EC axioms, the domain model consists of the causal laws of the domain, and the narrative provides observations of event occurrences and properties of fluents. Consider the following example from [12]:

Example 1 (Light on/off): A model of a light that can be turned on and off. First, we define the domain model (Fig. 1a) where: (i) the state of the light is represented by a fluent (line 1), (ii) turning it on/off is represented by events (line 2), (iii) event effects are stated at lines 3–4, and s(CASP) generates lines 6–7 (discussed in Section 2.1.1). Second, we define a narrative (Fig. 1b). The light is initially off (line 1) and it is turned on/off at times 10/20 (line 2). We query timepoints at which the light is on (line 3) and the answer is between 10 and 20. The code is available at ex1/model.pl.[1]

The relevant EC axioms for this example are shown in Fig. 1c and implemented in axioms/bec_scasp-small.pl. The predicate `holdsAt(F, T2)` is provable for time `T2` if an event *happened* at some earlier time `T1` whose effect was to *initiate* that fluent at `T1` and the fluent was *not stopped* between `T1` and `T2`. A fluent is stopped in a time interval if some event occurs within the interval that terminates it. The full 7 BEC axioms, used in further examples, are implemented in axioms/bec_scasp.pl. Implementation of the predicate `not_stoppedIn/3` is discussed in Section 2.1.1.

2.1.1 Modeling EC Axioms in s(CASP)

Two key factors contribute to the s(CASP)'s ability to model EC in continuous time: the preservation of non-ground variables during the execution and the integration with constraint solvers. Using the trans-

[1] All files are available at https://github.com/ovasicek/iclp25-scasp-zeno/tree/master/examples DOI 10.5281/zenodo.15737913.

lation rules, introduced in [3], one can translate the BEC axioms into s(CASP) programs that follow the logic programming convention. However, special considerations must be made to avoid non-termination.

First, it is crucial to consider the relative order of `happens/2` and `initiates/3` (same for `terminates/3` and `releases/3`) in the axiom for `holdsAt/2` shown in Fig. 1c. When proving `happens/2` first, the reasoning might go through an expensive (or non-terminating) proof of an event happening only to then realize that it has no effects on the fluent. However, proving `initiates/3` first can lead to the reasoning trying to prove the effect of an infinite number of events without considering whether they happen[2]. In [18], we proposed to define predicates `can_initiates/3`, `can_terminates/3`, `can_releases/3`, and `can_trajectory/4` via preprocessing that is done by s(CASP) automatically when using the `--ec` command-line option. It defines a fact `can_initiates(E,F,T)` for each fact and/or rule `initiates(E,F,T):- some_body` (and similarly for others) without duplicates. Custom rules can be defined manually. This has proven effective in avoiding non-termination while pruning the search space.

Second, s(CASP) implements the `not` keyword by generating *dual rules*. The generation is uninformed of the semantics of the predicates and, thus, generates more rules than needed including ones that may explore unrealistic combinations of values of continuous time arguments. We use a custom implementation of the predicates `not_stoppedIn(T1, F, T2)` and `not_startedIn/3` in order to avoid non-termination. The custom implementation (in axioms/bec_scasp.pl, line 169) leverages the semantics of the predicates to be more efficient and prevent non-termination.

3 Zeno-Like Behavior in s(CASP): Zeno-Descending Chain of Events

In our experiments, the main source of non-termination are *triggered events*. An event is triggered if its occurrences are implied by rules of the domain model. In continuous time, such trigger rules can lead to an infinite number of occurrences and, thus, to an infinite number of changes to reason about.

We have encountered many examples that share the same pattern in their reasoning trees while non-terminating. The reasoning gets stuck in a loop through event trigger rules, where each iteration has a `happens(Event, TimeN)` predicate with a new time variable which is strictly less than the one from the previous iteration but otherwise both are constrained to the same interval. The tree thus contains a chain: `happens(e,T1)`, `T2 < T1`, `happens(e,T2)`, ..., `TN < ... < T2 < T1`, `happens(e,TN)`. We call such a sequence a *Zeno-descending chain of events* (or Zeno chain). A crucial aspect is that there is no time delta between the `T`'s, and so the distance between the timepoints can be infinitesimal. Reasoning with a Zeno chain results in *exploring* an infinite number of event occurrences in a finite time interval, which matches the definition of Zeno behavior. However, as we show throughout this paper, a Zeno chain can be encountered by goal-directed reasoning even in examples in which only a finite number of events occur. Therefore, not all Zeno chains represent Zeno behavior but rather only *Zeno-like* behavior.

In Section 3.2, we propose a technique for automatic detection of a Zeno chain, which allows us to alert users to its presence and to halt the non-terminating reasoning.

3.1 Example(s) of a Zeno-Descending Chain of Events

Example 2 (Simplified bank account ↗): A *simplified version* of the bank account problem [12] reasons about a single bank account and money withdrawal from it. If the balance is below the minimum of 1000, then a service fee of 10 will be charged via a triggered event. In this version, the service fee can only be charged once *per narrative* (instead of once per month). Fig. 2 shows the s(CASP) encoding of this example and an example narrative with a few queries and their expected answers. The code is available at ex2/model.pl.

[2]Such non-termination is shown in ex1/logs/non_term_summary.log using a modified version of the light example.

```
fluent(balance(M)).   fluent(noServiceFeeYet).
event(withdraw(M)).   event(serviceFee).

initiates(withdraw(X), balance(NewB), T) :-
  NewB #= OldB - X,
  holdsAt(balance(OldB), T).
terminates(withdraw(_), balance(OldB), T) :-
  holdsAt(balance(OldB), T).

terminates(serviceFee, noServiceFeeYet, T).
initiates(serviceFee, balance(NewB), T) :-
  NewB #= OldB - 10,   % service fee is 10
  holdsAt(balance(OldB), T).
terminates(serviceFee, balance(OldB), T) :-
  holdsAt(balance(OldB), T).
```

(a) Domain model (ex2/model.pl).

```
happens(serviceFee, T) :-
  holdsAt(noServiceFeeYet, T),
  B #< 1000,   % min balance is 1000
  holdsAt(balance(B), T).
```

(b) Problematic event trigger rule.

```
initiallyP(balance(10000)).
initiallyP(noServiceFeeYet).
happens(withdraw(8000), 10).
happens(withdraw(1500), 20).

?- holdsAt(balance(X),  5). % 10000
?- holdsAt(balance(X),  15). % 2000
?- holdsAt(balance(X),  25). % 490
?- happens(serviceFee,  T).  % smallest T > 20
```

(c) Narrative and queries with expected answers.

Figure 2: s(CASP) encoding of Example 2 (simplified bank account).

```
1  happens(serviceFee, {T1~[T1 > 0, T1 =< 100]})                    % T1
2    holdsAt(noServiceFeeYet, {T1~[T1 > 0, T1 =< 100]})
3      initiallyP(noServiceFeeYet)
4      not_stoppedIn(0,noServiceFeeYet, {T1~[T1 > 0, T1 =< 100]})
5        {T2~[T2 > 0, T2 < 100]} #< {T1~[T1 > 0, T1 =< 100]}       % T2 < T1
6        happens(serviceFee, {T2~[T2 > 0, T2 < 100]})              % T2
7          holdsAt(noServiceFeeYet, {T2~[T2 > 0, T2 < 100]})
8            initiallyP(noServiceFeeYet)
9            not_stoppedIn(0,noServiceFeeYet, {T2~[T2 > 0, T2 < 100]})
10               {T3~[T3 > 0, T3 < 100]} #< {T2~[T2 > 0, T2 < 100]}  % T3 < T2
11               happens(serviceFee, {T3~[T3 > 0, T3 < 100]})        % T3
```

Figure 3: Zeno-descending chain of events in a simplified trace of Ex. 2 (full in ex2/logs/non_term.log)

All queries from Fig. 2c non-terminate due to the trigger rule of `serviceFee` (Fig. 2b). A simplified snapshot of the non-terminating reasoning tree is shown in Fig. 3. The Zeno chain starts at line 6 and continues at line 11. The reasoning enters a Zeno chain because to prove that the fee happened at some `T2`, it must consider that it could have already happened earlier at `T3`, but then it must consider an even earlier `T4`, etc. Details and solutions are discussed in Section 4.

In the following sections, we present and describe solutions to a selected class of problems that lead to non-termination through a Zeno chain. Each class of problems is introduced with an example from the literature or a variation thereof. Table 1 summarizes all the examples, alongside further unnumbered examples (available at the repository under the folder named other), and for each example shows which combination of problems (Zeno-like or Zeno behavior) is present.

3.2 Automated Detection of Zeno-Descending Chains of Events

We propose a simple yet effective technique to detect Zeno chains, made available as a feature of s(CASP) from version 0.25.06.25 at https://gitlab.software.imdea.org/ciao-lang/sCASP. It can be enabled using the `--zeno_halt` flag. When a Zeno chain is detected, s(CASP) stops and issues a warning to inform the user (avoiding non-termination). This technique, works as follows: when a predicate `happens(E,T)` is expanded, s(CASP) will check the current derivation looking for predicates `happens(E,T1)` and `happens(E,T2)`, such as: (i) all three have the same event `E`, the variables `T1` and `T2` are constrained to the same interval (`low < Tx < high`), (ii) `happens(E,T1)` was expanded earlier than `happens(E,T2)`, and (iii) `T2` is strictly smaller than `T1`, i.e., there is a constraint `T2 < T1`. E.g., the Zeno chain shown in Fig. 3 between lines 6 and 11 would be detected (see the full detection trace in ex2/logs/zeno_halt.log).

We validated the ability of this technique to (quickly) detect Zeno-like behaviors with each of the examples listed in Table 1 (their detection traces are available in the repository).

Table 1: Summary of examples and their problem composition.

		Repeated trigger	Trajectory related problems			Zeno Behaviors	
			Self-ending	Circular	With inequal.	Trivial	Paradoxes
Light on/off	Ex. 1						
Simpl. bank account	Ex. 2	*					
Bank account		*					
Light trigger off		*					
Fading light	Ex. 3		*				
Train gate			*				
Falling object			*				
Filling vessel			*				
Pulsing light	Ex. 4		*	*			
No-loss water tanks			*	*			
No-loss bouncing ball			*	*			
Simpl. water tanks	Ex. 5	*	*		*		
Blinking light	Ex. A1	*				*	
Water tanks	Ex. 6	*	*	*	*		*
Bouncing ball			*	*			*

4 Preventing Repeated Triggering of Events

A common concept in EC is preventing events from triggering repeatedly [12]. However, reasoning in continuous time introduces issues. Consider Ex. 2 from the previous section. A service fee should be charged if two conditions are met: (i) the balance is below some minimum value and (ii) the fee has not been charged yet. When money is withdrawn at time `T1`, the new balance will be provable via `holdsAt(ballance(X),T2)` for any `T2` such that `T2 > T1` (recall the axiom from Fig. 1c). Should the withdrawal make the balance go below the minimum, Condition (i) would hold for all `T2`'s (a continuous interval). Without Condition (ii), the fee would be charged infinitely many times. With Condition (ii), the first occurrence *prevents any subsequent* ones. However, there is a problem when we consider continuous time—it is impossible to find the first occurrence due to the non-inclusive lower bound, in our case `T2` such that `T2 > T1`. As a consequence, it is inherently impossible to answer the query from Fig 2c considering continuous time, causing non-termination in s(CASP) due to its attempt to find the smallest `T2` (recall Fig. 3). To solve this problem we propose two solutions:

Solution 1 (Add Epsilon): An epsilon in time is introduced and the fee is charged after a delay (see Fig. 4a). This is achieved using a new predicate `holdsAt(F, T, EPS, E)` (see its encoding in Fig. 4b). The third argument makes the predicate provable only at time `T = T1 + EPS` where `T1` is the timepoint of occurrence of the event `E`, specified by the fourth argument, that initiates the fluent `F`. The fourth argument is not strictly needed but helps avoid a slowdown from exploring irrelevant events. This solution ensures that the trigger rule proves `happens(withdraw(X), T1)` first and only then triggers the `serviceFee` at some `T2 = T1 + EPS`. Since `withdraw(X)` is not a triggered event, there only is a finite number of `T1`'s to explore. Hence, non-termination through this rule is no longer possible and the query terminates (see ex2/logs/fix-holdsAt4.log). Introducing an epsilon here is not like discretizing since `T1` and `T2` can still take any value in the continuous domain (and there is no impact on reasoning time).

Solution 2 (Remodel Behavior): We remodel the example with continuous time in mind to charge the fee at the same time as the withdrawal and combine the effects of the two events (see Fig. 4c). The fee is charged at the originally lower bound instead of infinitely close to it, and the query terminates (see ex2/logs/fix-remodel.log). In this example, the effect of the `serviceFee` is to initiate a new balance by combining both the amount subtracted by the fee and the event which caused it. The `withdraw(X)` event has no effect if the fee is charged at the same time. This solution is closest to the original behavior, but it is only suitable for examples where the effects of the events can be combined.

```
happens(serviceFee, T2) :-
  EPS #= 1/1000000,   B #< 1000,
  holdsAt(balance(B), T2, EPS, withdraw(_)),
  holdsAt(noServiceFeeYet, T2).
```

(a) Solution 1: Add Epsilon (ex2/fix-holdsAt4.pl).

```
holdsAt(Fluent, T2, Dur, Event) :-
  T2 #= T1 + Dur, ..., happens(Event, T1),
  initiates(Event, Fluent, T1), ...
```

(b) New axiom (axioms/bec_scasp.pl, line 118).

```
happens(serviceFee, T) :- happens(withdraw(Amount), T),
  holdsAt(noServiceFeeYet, T),  NewB #< 1000,
  NewB #= OldB - Amount, holdsAt(balance(OldB), T).
initiates(withdraw(X), balance(NewB), T) :-
  not_happens(serviceFee, T), ...
terminates(withdraw(_), balance(OldB), T) :-
  not_happens(serviceFee, T), ...
initiates(serviceFee, balance(NewB), T) :-
  happens(withdraw(X), T),
  NewB #= (OldB - X) - 10, holdsAt(balance(OldB), T).
```

(c) Solution 2: Remodel Behavior (ex2/fix-remodel.pl).

Figure 4: Two solutions to the problem in Example 2 (simplified bank account).

Using the above solutions, we modeled additional examples that feature the same problem: (i) the **full bank account** models multiple accounts and transactions and adds a monthly reset for the service fee (encoding in other/1-bank_account). (ii) the **light trigger off**, a version of Ex. 1 where the light automatically turns off when it is on (encoding in other/2-light_trigger_off).

5 Trajectory-Related Problems

EC represents continuous change via *trajectories*. A trajectory, defined using `trajectory(F1,T1,F2,T2):-body`, starts when its *control fluent* `F1` is initiated at `T1` by an event and stays active until the fluent is terminated. While the trajectory is active, it is used to determine the value of the *continuous fluent* `F2` at any `T2`. Below, we discuss three non-termination problems with trajectories that we have encountered.

5.1 Self-Ending Trajectory Problem

Autotermination is a common modeling pattern where a trajectory ends via an event triggered based on the continuous value defined by the trajectory. We call such trajectories *self-ending*. Modeling these in s(CASP) can lead to non-termination. Consider the following example:

Example 3 (Fading light, cont. Ex. 1): When the light is turned on, its brightness gradually fades in from 0 until it reaches full brightness of 10. The continuous change of the fluent `brightness(X)` is modeled by a trajectory controlled by the fluent `fading_in` and ends automatically via a triggered event `fade_in_end`. The event triggers when full brightness is reached and initiates `brightness(10)` as the new discrete value of the fluent. Fig. 5 shows the new parts of the s(CASP) encoding (compared to Ex. 1) and an example narrative with a few queries. The code is available at ex3/model.pl.

All queries in Fig. 5c non-terminate, looping through the trigger rule (Fig. 5a, line 13) and the trajectory axiom (Fig. 5b, lines 1-4). The Zeno chain looks like this: `happens(fade_in_end,T1)`, `holdsAt(brightness(10),T1)`, `T2<T1`, `can_initiates(fade_in_end,brightness(10),T2)`, `happens(fade_in_end,T2)` (see ex3/logs/non_term_summary.log). This chain can be detected automatically (see ex3/logs/zeno_halt.log). The problem is that the effect of the event is considered as a way to prove its own trigger. This is a limitation of goal-directed reasoning as the example does not inherently contain any Zeno(-like) behavior. We propose one solution:

Solution 1 (Restrict HoldsAt): When reasoning about ending a trajectory, we assume that it is active; otherwise we would not need to reason about its end. We restrict that `holdsAt(brightness(10),T)` should only be proven via the trajectory controlled by the fluent `fading_in` (see Fig. 6a) using a new predicate `holdsAt(F,T,CF)` (Fig. 6b), which limits the choice of control fluents when proving the value of a fluent. This prevents the original loop, and the query terminates (see ex3/logs/fix-holdsAt3.log).

```
event(fade_in_end).   fluent(light_on).
fluent(brightness(X)).   fluent(fading_in).
initiates(turn_light_on, fading_in, T).
releases(turn_light_on, brightness(X), T).
terminates(fade_in_end, fading_in, T).
initiates(fade_in_end, brightness(10), T).
terminates(fade_in_end, brightness(X), T):- X #≠ 10.

trajectory(fading_in, T1, brightness(NewB), T2) :-
  NewB #= OldB + ((T2-T1) * 1),
  holdsAt(brightness(OldB), T1).

happens(fade_in_end, T) :- holdsAt(brightness(10), T).
```

(a) Domain model additions (ex3/model.pl).

```
holdsAt(Fluent2, T2):-  T1 #> 0,  T1 #< T2,
  can_trajectory(Fluent1, T1, Fluent2, T2),
  can_initiates(Event, Fluent1, T1),
  happens(Event, T1),
  initiates(Event, Fluent1, T1),
  trajectory(Fluent1, T1, Fluent2, T2),
  not_stoppedIn(T1, Fluent1, T2).
```

(b) Trajectory axiom (axioms/bec_scasp.pl).

```
initiallyP(brightness(0)).
happens(turn_light_on,    10).
?- holdsAt(brightness(X), 25). % 10
?- happens(fade_in_end,   T).  % 20
```

(c) Narrative and some queries.

Figure 5: s(CASP) encoding of Example 3 (fading light), extension of Example 1 (light on/off)

```
happens(fade_in_end, T) :-
  holdsAt(brightness(10), T, fading_in).
```

(a) Solution: Restrict HoldsAt (ex3/fix-holdsAt3.pl).

```
holdsAt(Fluent2, T2, Fluent1) :- ...,
  trajectory(Fluent1, T1, Fluent2, T2), ...
```

(b) New axiom (axioms/bec_scasp.pl, line 51).

Figure 6: Solution for the problem in Example 3 (fading light).

The above problem is present in all the trajectory-related examples in the rest of this paper. The solution from this section is applied in all of them by default. In addition, we have modeled the following examples that feature the same problem: (i) the **filling vessel** [16] which stops filling and starts spilling once it fills up to its maximum level (encoding in other/5-filling_vessel/). (ii) The **train gate** controller [1] models a railway crossing with trajectories for lowering/raising the crossing gates (encoding in other/3-train_gate/). (iii) The **falling object** example [12] models an object that is dropped from some height and falls to the ground (encoding in other/4-falling_object/).

5.2 Circular Trajectory Problem

This section introduces a problem of *circular dependency* between trajectories—the end of one starts the other and vice versa. The following example does not inherently contain any Zeno(-like) behavior; however, it is a problem for goal-directed, top-down reasoning in continuous time.

Example 4 (Pulsing light, cont. Ex. 3): The light fades in and out repeatedly. As soon as the light reaches maximum brightness, it will start fading out back to zero. Once zero is reached, the light will start fading in again, repeating the process. Fig. 7 shows the modification of the s(CASP) encoding (compared to Ex. 3) and a narrative with a few queries. The code is available at ex4/model.pl.

All the queries from Fig. 7b non-terminate. To prove the effect of a trajectory at some `T2`, we first need to find its start time `T1` (see the axiom in Fig. 5b). However, due to the circularity, the reasoning will proceed as follows: (i) try to prove the effect of trajectory `A` at `TA2`, (ii) need to prove its start time `TA1` such that `TA1 < TA2`, (iii) trajectory `A` can start via the end of an earlier trajectory `B`, (iv) need to prove the effect of trajectory `B` at `TB2` such that `TB2 = TA1`, which is Step (i) with `TB2 < TA2`. The loop contains the trajectory axiom (Fig. 5b lines 1–4) and the trigger rules of the trajectory start/end events (Fig. 7a lines 6–7, Fig. 6a lines 1-2). The Zeno chain looks as follows: `happens(fade_out_end,T1)`, `holdsAt(brightness(0),T1)`, `T2<T1`, `happens(fade_in_end,T2)`, `holdsAt(brightness(10),T2)`, `happens(fade_out_end,T3)` (see ex4/logs/non_term_summary.log); and it can be automatically detected (see ex4/logs/zeno_halt.log). The problem is that the reasoning never reaches the body of the `trajectory/4` predicate which means that there is no knowledge of the potential duration of the trajectory, i.e., the reasoning is exploring an infinite number of infinitely short trajectories. We propose two solutions for this problem:

```
fluent(fading_out).          event(fade_out_end).
terminates(fade_out_end, fading_out, T).
initiates(fade_out_end, fading_in, T).
trajectory(fading_out, T1, brightness(NewB), T2) :-
  NewB #= OldB - (T2-T1), holdsAt(brightness(OldB), T1).
happens(fade_out_end, T) :-
  holdsAt(brightness(0), T, fading_out).
```

(a) Encoding additions (ex4/model.pl).

```
initiallyP(brightness(0)).
happens(turn_light_on,   10).

?- happens(fade_in_end,  T). % 20
                             % and 40
?- happens(fade_out_end, T). % 30
```

(b) Narrative and some queries.

Figure 7: s(CASP) encoding of Example 4 (pulsing light), extension of Example 3 (fading light).

```
happens(fade_in_end, T) :-   Dur #= 10,
  holdsAt(brightness(10), T, fading_in, Dur).
happens(fade_out_end, T) :-   Dur #= 10,
  holdsAt(brightness(0), T, fading_out, Dur).
```

(a) Solution 1: Add Duration (ex4/fix-holdsAt4.pl).

```
holdsAt(Fluent2, T2, Fluent1, Duration) :-
  T2 #= T1 + Duration, ...
  trajectory(Fluent1, T1, Fluent2, T2), ...
```

(b) New axiom (axioms/bec_scasp.pl, line 65).

```
incr_event(fade_in_end).
incr_event(fade_out_end).

can_initiates(fade_in_end, fading_out, T) :-
  incr_happens(fade_in_end, T).

can_initiates(fade_out_end, fading_in, T) :-
  incr_happens(fade_out_end, T).
```

(c) Solution 2: Incremental Reasoning (ex4/fix-incr.pl).

Figure 8: Two solutions for the problem in Example 4 (pulsing light).

Solution 1 (Add Duration): Introduces duration information into the event trigger rules (see Fig. 8a) using a new predicate `holdsAt(F1,T,F2,Duration)` (see Fig. 8b) which is based on the `holdsAt/3` predicate from Ex. 3 and extended with a duration similarly to the `holdsAt/4` predicate from Ex. 2. It specifies a trajectory control fluent `F2` and a `Duration` for which the fluent must hold. This introduces the duration of the trajectories into the loop and the query terminates (see ex4/logs/fix-holdsAt4.log). This approach is suitable for cases where the trajectories always have the same duration; for cases with variable duration, we could specify a minimum duration or use the next solution.

Solution 2 (Incremental Reasoning): Uses forward reasoning for trajectories and accurately deals with trajectories with variable duration. The reasoning starts with the *first trajectory*. Once we prove its end, we can start reasoning whether it caused the start of some second trajectory and so on. This is achieved by taking the trajectory ending events (Fig. 8c, lines 1-2) and decoupling all their effects, other than terminating the trajectory, using `incr_happens(E, T)` (Fig. 8c, lines 4-8). The reasoning repeatedly executes queries to see if any of the events happen. If an occurrence of `E1` is found at `T1`, then a fact `incr_happens(E1, T1)` is added to the knowledge base. The next increment will then query events at times greater than `T1`. If multiple occurrences are found in an increment, only the earliest ones are saved as they could impact the occurrence of the later ones in the next increment. This process is repeated until no more occurrences are found or until a time limit is reached. The original query is then executed (see ex4/logs/fix-incr.log) with the knowledge base of occurrences of all incremental events. This approach is implemented in s(CASP) and can be enabled via the `--incremental` cmd-line option.

Using the above, we have modeled two additional examples: (i) The **no-loss bouncing ball** is a simplified bouncing ball that bounces up to the original height on every bounce (encoding in other/7-no_loss_bouncing_ball). (ii) The **no-loss water tanks** models two water tanks which are being slowly drained and one at a time is being refilled by a water input arm, while only considering narratives where the combined draining rate is the same as the input rate (encoding in other/6-no_loss_water_tanks).

5.3 Self-Ending Trajectory with End Condition Based on Inequality

In the previous trajectory examples, we considered trajectories whose end events were triggered based on equality, e.g., the light stops fading in when its brightness is *equal* to 10. In the case of strictly monotonic

```
fluent(water_left(X)).
fluent(left_filling).   fluent(left_draining).
event(switch_left).   event(switch_right).

initiates(switch_left, left_filling, T).
terminates(switch_left, left_draining, T).
terminates(switch_right, left_filling, T).
initiates(switch_right, left_draining, T).

trajectory(left_filling, T1, water_left(NewW), T2) :-
  NewW #= OldW + ((T2-T1) * 10), % rate 30 - 20
  holdsAt(water_left(OldW), T1).
trajectory(left_draining, T1, water_left(NewW), T2) :-
  NewW #= OldW - ((T2-T1) * 20), % out rate 20
  holdsAt(water_left(OldW), T1).

happens(switch_left, T):- W #=< 50, % target lvl 50
  holdsAt(water_left(W), T, left_draining).
```

(a) Partial domain model (full at ex5/model.pl).

```
initiallyP(water_left(100)).
happens(start(right),     10).
happens(switch_right,     16.25).

?- happens(switch_left,   T). % 12.5, 18.125
?- holdsAt(water_left(X), 12.5).  % 50
?- holdsAt(water_left(X), 16.25). % 87.5
?- holdsAt(water_left(X), 18.125).% 50
?- holdsAt(water_left(X), 19.5).  % 50.625
```

(b) Working narrative (ex5/model-nar1.pl).

```
initiallyP(water_left(0)).
happens(start(left),      10).
happens(switch_right,     13).

?- holdsAt(water_left(X), 13).% 30 (term.)
?- happens(switch_left, T).   % smallest T>13
?- holdsAt(water_left(X), 15).% 50
```

(c) Non-term. narrative (ex5/model-nar2.pl).

Figure 9: s(CASP) encoding of Example 5 (simplified water tanks).

```
happens(switch_left, T) :- W #= 50,  % = target lvl
  holdsAt(water_left(W), T, left_draining).
happens(switch_left, T) :-   Dur #= 1,
  W #< 50,   % < target lvl
  holdsAt(water_left(W), T, left_draining, Dur).
```

(a) Solution 1: Add Duration (ex5/fix-split_holdsAt4.pl).

```
terminates(switch_right, left_filling, T) :-
  X #> 50, % > target level 50
  holdsAt(water_left(X), T, left_filling).
initiates(switch_right, left_draining, T) :-
  X #> 50, % > target level 50
  holdsAt(water_left(X), T, left_filling).
```

(b) Solution 2: Remodel Beh. (ex5/fix-split_no_start.pl).

Figure 10: Two solutions for the problem in Example 5 (simplified water tanks).

trajectories, the brightness will be exactly 10 only at one timepoint, and so the end event can only occur once. However, sometimes an end condition with inequality may be called for. In such cases, we can face a manifestation of the repeated trigger problem from Ex. 2 where an event was defined to occur infinitely close to some non-inclusive bound. Consider the following example:

Example 5 (Simplified water tanks ↗): A simplified version of the well-known example of two leaking water tanks [2], where the tanks share one input arm that switches back and forth between them in an attempt to keep their water levels above a target level. Our simplified version only models the left tank, and any switches to the right tank come only as facts in the narrative. There is a trajectory for draining the left tank via a draining rate (20 units) and a second trajectory for filling it via a combination of an input rate (30 units) and the draining rate. A triggered event `switch_left` causes the input arm to switch to the left tank. It is triggered while the tank is draining if the current water level is *equal to or below* the target level (50 units)—the draining trajectory has an end condition with inequality. Fig. 9 shows most of the s(CASP) encoding of this example and two example narratives with a few queries. The code is available at ex5/model.pl.

Queries for the first narrative (Fig. 9b) all terminate.[3] However, the second and third query in the second narrative (Fig. 9c) do not. In the first narrative, the arm starts in the right tank meaning that the tank is initially draining from its starting level 100, drops to the target level 50 at time 12.5, and a `switch_left` is triggered. A `switch_right` happens at time 16.25, when the tank is already filled up to 87.5 (above the target level). In the second narrative, the tank starts empty and is filling. A `switch_right` happens at time 13 while the left tank is only at level 30—still below the target level. The filling trajectory ends at time 13, but a `switch_left` is triggered as soon as possible at some `T > 13`. However, this is a problem

[3] However, a more complex impl. of `not_stoppedIn/3` is needed (in axioms/bec_scasp-interval_not.pl, line 194).

because an infinitely short draining trajectory is created. The reasoning non-terminates due to its attempt to find an event infinitely close to a non-inclusive lower bound (see ex5/logs/non_term_summary.log and detection in ex5/logs/zeno_halt.log). Such an event cannot be found in continuous time, hence, it is impossible to answer the query. This is a manifestation of the repeated trigger problem from Ex. 2 in the context of a trajectory. We propose two solutions, similar to the ones from Ex. 2. Both solutions split the reasoning into two situations: (a) the trajectory starts when its end condition is not met yet (1st narrative) and (b) the trajectory starts when its end condition already is met (2nd narrative). In Case (a), the trajectory reaches the equality first and ends before the inequality becomes applicable , therefore, the inequality is not needed. Similarly, in Case (b), the equality is never reached (it has been already exceeded) and hence not needed.

Solution 1 (Add Duration): Uses an equality-based trigger condition for Case (a) and introduces a minimum duration for the trajectory in Case (b) using the `holdsAt/4` predicate from Ex. 4 (see Fig. 10a). The fixed duration prevents the occurrence of an infinitely short trajectory, removing the non-termination problem (see ex5/logs/fix-split_holdsAt4.log). The duration is inherently required by this example and does not lead to the inaccuracies that would arise if we discretized.

Solution 2 (Remodel Behavior): Also uses an equality-based trigger condition for Case (a) but a different approach for Case (b). An infinitely short trajectory only makes an infinitely small difference, thus, should not start at all. This can be achieved by limiting all the effects of the `switch_right` event only to Case (a) (see Fig. 10b) and the query terminates (see ex5/logs/fix-split_no_start.log).

6 True Zeno Behavior

We say the prior examples contain *Zeno-like* behaviors as they do not fully meet the definition of Zeno behavior—the reasoning was *considering* an infinite number of events in finite time, but they never *needed to happen*. This section presents a true Zeno behavior in the **water tanks** [2] where an infinite number of events occurs due to delays between them getting shorter and shorter. In addition, we modeled two more examples: (i) the **bouncing ball** [9] which looses energy with each bounce. (ii) The **blinking light** which blinks with an infinitesimal delay, showing trivial Zeno behavior (detailed in A).

Example 6 (Water tanks, cont. Ex. 5): The full version of the water tanks [2], where the tanks and the arm switches are modeled via triggered events based on reasoning about both water levels. The Zeno behavior [2, 20, 9] manifests when the input flow rate is higher than the output flow rates of each tank but lower than their combined output rates—the arm is able to fill each tank, but overall the amount of water in the system is decreasing. The arm switches instantaneously and, thus, will keep both tanks from going below the target level by switching for shorter and shorter durations as they both drop closer to their target levels. Fig. 11 sketches s(CASP) encoding (full code in ex6/model.pl).

Queries shown in Fig. 11b non-terminate (see ex6/logs/non_term_summary.log) due to a combination of the problems that we have presented so far: (a) real Zeno behavior, (b) the repeated trigger problem from Ex. 2 manifested in a trajectory, (c) the self-ending trajectory problem from Ex. 3, (d) the circular trajectory problem from Ex. 4, and (e) the inequality end condition problem from Ex. 5. All the sub-problems, except (a), cause non-termination through a Zeno chain and, thus, can be detected (ex6/logs/zeno_halt.log). Like the two prior examples, the initial encoding already solves Problem (c) via the `holdsAt/3` predicate. We further propose three solutions:

Partial Solution 1 (Incremental Reasoning): This solution addresses Problem (d), which is due to reasoning limitations, and terminates up to Problem (a). We solve Problem (d) via incremental reasoning (see Fig. 12a). The incremental events are the events for switching the arm left and right. Due to the

```
fluent(right_filling). % renamed fluent(left_draining)
fluent(water_right(X)).

trajectory(right_filling, T1, water_right(NewW), T2) :-
  ... % similar to trajectory for water_left
trajectory(right_filling, T1, water_right(NewW), T2) :-
  ... % similar to trajectory for water_left

happens(switch_right, T) :-
  CurrW #=< 50,   % target level
  holdsAt(water_right(CurrW), T, left_filling).
```

(a) Domain model additions (ex6/model.pl).

```
initiallyP(water_left(100)).
initiallyP(water_right(100)).
happens(start(right),      10).
?- T #=< 19.5,
   happens(switch_left, T). % 12.5, 18.125
?- T #=< 19.5,
   happens(switch_right, T).% 16.25, 19.0625
?- holdsAt(water_left(X),  12.5).  % 50
?- holdsAt(water_right(X), 16.25). % 50
?- holdsAt(water_right(X), 19.5).  % 54.375
?- holdsAt(water_left(X),  19.5).  % 50.625
```

(b) Narrative and some queries.

Figure 11: s(CASP) encoding of Example 6 (water tanks), full version of Example 5.

```
incr_event(switch_left).   incr_event(switch_right).
can_initiates(switch_left, left_filling, T) :-
  incr_happens(switch_left, T).
can_initiates(switch_right, right_filling, T) :-
  incr_happens(switch_right, T).
?- !incr_max_time(19.5), ...
```

(a) Partial Solution 1: Incremental (ex6/partfix-incr.pl).

```
happens(switch_right, T) :-    MinD #= 1,
  D #>= MinD,    W #=< 50,   % target level 50
  holdsAt(water_right(W), T, left_filling, D).
% same for happens(switch_left, T) :- ...
```

(b) Solution 2: Add Duration(ex6/fix-holdsAt4.pl).

```
% same as Solution 1
incr_event(...).
can_initiates(...) :- incr_happens(...).

% similar to Sol 2., but split trigger rules
happens(switch_right, T) :-   MinD #= 1,
  D #>= MinD,   W #= 50,   % target level 50
  holdsAt(water_right(W), T, left_filling, D).
happens(switch_right, T) :-   D #= 1,
  W #< 50,   % target level 50
  holdsAt(water_right(W), T, left_filling, D).
% same for happens(switch_left, T) :- ...
```

(c) Solution 3: Combined (ex6/fix-incr_holdsAt4.pl).

Figure 12: Three solutions for the problem in Example 6 (water tanks).

Zeno behavior, typical narratives will not run into Problems (e) and (b) as the tanks never go below the target level (cf. ex6/logs/partfix-incr.log).

Solution 2 (Add Duration): This solution addresses the Zeno-like behavior of Problems (b) and (e), and the Zeno behavior of Problem (a). Problem (b) manifests as part of Problem (e), and so we only have to focus on the latter. We solve Problem (e) by defining a *minimum* duration for the trajectories using the `holdsAt/4` predicate (see Fig. 12b). This introduces a delay between the switch events which, additionally, solves Problem (a)—highlighting the similarity of Zeno and Zeno-like behaviors. Moreover, solving Problem (e) via a minimum duration is also an alternative solution to Problem (d), meaning that incremental reasoning is no longer necessary for termination (see ex6/logs/fix-holdsAt4.log). However, a smaller value of the minimum negatively impacts reasoning time as it allows more iterations of the loop.

Solution 3 (Combined): This solution combines Solutions 1 and 2 (see Fig. 12c and ex6/logs/fix-incr_holdsAt4.log). While Solution 1 is only a partial solution, the combination of both solutions leads to better performance (from 30 min to 10 seconds).

Note that other Zeno behaviors may require still other solutions. For example, a minimum duration for the trajectory of a **bouncing ball** results in it falling through the floor. In this case, a suitable solution is to make it stop bouncing when the last bounce was shorter than a minimum duration or to introduce a flat loss of velocity on each bounce (modeled in other/8-bouncing_ball).

7 Conclusion and Summary

We have implemented a novel technique, made available as a feature of s(CASP), that automatically detects Zeno-like behaviors based on the presence of a Zeno-descending chain of events. We have further identified and characterized a set of non-termination problems, caused by Zeno-like behaviors, that can be encountered in common Event Calculus modeling tasks in continuous time and have proposed four

Table 2: Summary of the proposed solutions and their applicability to the presented problems.

		Repeated trigger	Trajectory related problems			Zeno Behaviors	
			Self-ending	Circular	With inequal.	Trivial	Paradoxes
Automated Detection	(Sect. 3.2)	✓	✓	✓	✓	✓	
Restrict HoldsAt	(Ex. 3, 4, 5, 6)		✓				
Incremental Reasoning	(Ex. 4, 6)			✓			
Add Epsilon / Duration	(Ex. 2, 4, 5, 6, A1)	✓		✓	✓	✓	✓
Remodel Behavior	(Ex. 2, 5)	✓			✓		✓

types of solutions that can be used to solve them without discretizing. We believe that this set of problems (and solutions) is enough to cover a wide range of real use cases, such as a safety-critical system [18].

As shown in Table 2, the detection technique works for all classes of problems, except for nontrivial Zeno behaviors because those feature a descending chain of events with delays between them getting shorter and shorter (instead of infinitesimally small). Additionally, it summarizes the proposed solutions, in which examples they have been applied, and which classes of problems are solved by each of them. The **Restrict HoldsAt** solution uses external knowledge of the example to guide EC reasoning by restricting the search rule to chose a specific clause of the predicate `holdsAt`. **Incremental Reasoning** addresses circularity via forward reasoning which is particularly suitable for EC due to its need to reconstruct history. The **Add Epsilon / Duration** solution introduces a time epsilon into the loop, replacing infinitesimal time steps without causing inaccurate behaviors (as would be the case if we discretized). The **Remodel Behavior** solution tunes the model of the example for reasoning in continuous time to avoid infinitesimal time steps while preserving the intended behavior.

References

[1] Rajeev Alur & David L. Dill (1994): *A Theory of Timed Automata*. *TCS* 126, pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[2] Rajeev Alur & Thomas A. Henzinger (1997): *Modularity for Timed and Hybrid Systems*. In: *CONCUR*, *LNCS* 1243, Springer, pp. 74–88, doi:10.1007/3-540-63141-0_6.

[3] Joaquín Arias, Manuel Carro, Zhuo Chen & Gopal Gupta (2022): *Modeling and Reasoning in Event Calculus using Goal-Directed Constraint Answer Set Programming*. *TPLP* 22(1), pp. 51–80, doi:10.1017/S1471068421000156.

[4] Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple & Gopal Gupta (2018): *Constraint Answer Set Programming without Grounding*. *TPLP* 18(3-4), pp. 337–354, doi:10.1017/S1471068418000285.

[5] Michael Gelfond & Vladimir Lifschitz: *The Stable Model Semantics for Logic Programming*. In: *ICLP*, pp. 1070–1080, doi:10.2307/2275201.

[6] Rodolfo Gómez & Howard Bowman (2007): *Efficient Detection of Zeno Runs in Timed Automata*. In: *FORMATS*, *LNCS* 4763, Springer, pp. 195–210, doi:10.1007/978-3-540-75454-1_15.

[7] Thomas A. Henzinger (1996): *The Theory of Hybrid Automata*. In: *LICS*, IEEE CS, pp. 278–292, doi:10.1109/LICS.1996.561342.

[8] Frédéric Herbreteau & B Srivathsan (2013): *Coarse abstractions make Zeno behaviours difficult to detect*. *Logical Methods in Computer Science* Volume 9, Issue 1:6, doi:10.2168/LMCS-9(1:6)2013.

[9] Karl Henrik Johansson, Magnus Egerstedt, John Lygeros & Shankar Sastry (1999): *On the regularization of Zeno hybrid automata*. *Systems & control letters* 38(3), pp. 141–150, doi:10.1016/S0167-6911(99)00059-6.

[10] Andrew G. Lamperski & Aaron D. Ames (2013): *Lyapunov Theory for Zeno Stability*. *IEEE TAC* 58, pp. 100–112, doi:10.1109/TAC.2012.2208292.

[11] Erik T. Mueller (2008): *Chapter 17 Event Calculus*. In: *Handbook of Knowledge Representation, Foundations of Artificial Intelligence* 3, Elsevier, pp. 671–708, doi:10.1016/S1574-6526(07)03017-9.

[12] Erik T. Mueller (2014): *Commonsense Reasoning: An Event Calculus Based Approach*. Morgan Kaufmann, doi:10.1016/B978-0-12-801416-5.00002-4.

[13] Anitha Murugesan, Isaac Hong Wong, Joaquín Arias & et. al. (2024): *Automating Semantic Analysis of System Assurance Cases Using Goal-Directed ASP*. *TPLP* 24(4), pp. 805–824, doi:10.1017/S1471068424000425.

[14] Jonas Rinast & Sibylle Schupp (2012): *Static Detection of Zeno Runs in UPPAAL Networks Based on Synchronization Matrices and Two Data-Variable Heuristics*. In: *FORMATS*, Springer, Berlin, Heidelberg, pp. 220–235, doi:10.1007/978-3-642-33365-1_16.

[15] Murray Shanahan (1996): *Robotics and the Common Sense Informatic Situation*. In: *ECAI*, Wiley, pp. 684–688.

[16] Murray Shanahan (1997): *Solving the frame problem: a mathematical investigation of the common sense law of inertia*. MIT Press.

[17] Sarat Chandra Varanasi, Joaquín Arias, Elmer Salazar & et. al. (2022): *Modeling and Verification of Real-Time Systems with the Event Calculus and s(CASP)*. In: *PADL*, Springer International, doi:10.1007/978-3-030-94479-7_12.

[18] Ondrej Vasicek, Joaquin Arias, Jan Fiedor & et. al. (2024): *Early Validation of High-Level System Requirements with Event Calculus and Answer Set Programming*. *TPLP* 24(4), pp. 844–862, doi:10.1017/S1471068424000280.

[19] Don Ward (2012): *AVSI's System Architecture Virtual Integration Program: Moving SAVI to the Launch Pad*. In: *15th Annual NDIA Systems Engineering Conference*. Available at https://ndia.dtic.mil/wp-content/uploads/2012/system/ttrack914871.pdf.

[20] Jun Zhang, Karl Henrik Johansson & et. al. (2001): *Zeno hybrid systems*. *IJRNC* 11(5), pp. 435–451, doi:10.1002/rnc.592.

A Trivial Zeno Behavior: Zero Delay Circular Events

This appendix presents an example that did not quite fit into the paper. The example shows trivial Zeno behavior but it is a modeling pattern that is easily encountered in EC. When reasoning in continuous time, it is easy to introduce event trigger rules which will cause an infinite number of events to occur in a finite time interval. Events have no duration and their effects are instantaneous, unless we explicitly introduce some duration or delay. Consider the following example:

Example A1 (Blinking light, cont. Ex. 1): Both the events for turning the light on and off were changed into triggered events. The trigger rules define that the light should be turned off when it is on and vice versa. However, the encoding does not define a frequency for the blinking. The triggered events are defined to turn the light off/on *as soon as* it is on/off. Fig. 13 shows the encoding additions for this example (compared to Ex. 1) and an example narrative with a few queries. The code is available at apx1/model.pl.

All the queries in Fig. 13b will non-terminate and, furthermore, their expected answers are unknown. There are two problems: (i) two instances of the repeated trigger problem (from Ex. 2) one for each triggered event and (ii) trivial zeno behavior since the light is defined to blink infinitely quickly. Problem (ii) means that while the light is blinking we are unable to conclude whether it is on or off. The problem manifests as a circular dependency between the two triggered events without any delay in between (see apx1/logs/non_term_summary.log) and it can be automatically detected (see apx1/logs/zeno_halt.log).

We present our solution in Fig. 14. To address Problem (i) we utilize a simplified version of the first solution from Ex. 2. The second one is not suitable here because the turn on/off events have opposite effects. Solving Problem (i) leads to Problem (ii) being solved as well.

The solution (Fig. 14a) adds a delay to the event trigger rules using a new predicate `holdsAt/3` (Fig 14b), which is the same as the `holdsAt/4` introduced in Ex. 2 except without the fourth argument, which is not needed here (there is only one event to consider for each rule).

To address Problem (ii), the Zeno behavior, we need to define some blinking frequency for the light. Notice that the solution to Problem (i) already introduced a delay between the events and, thus, it solves this problem as well. This shows the similarity between Zeno behaviors and the Zeno-like behaviors that we have presented in this paper.

```
1 happens(turn_light_off, T) :-
2   holdsAt(light_on, T).
3 happens(turn_light_on, T) :-
4   not_holdsAt(light_on, T).
```

(a) Domain model additions (apx1/model.pl).

```
happens(turn_light_on,     10).
?-     holdsAt(light_on,   15). % ? (unknown)
?- happens(turn_light_off, T). % ? (unknown)
?- happens(turn_light_on,  T). % 10 and ? (unknown)
```

(b) Narrative and queries with unknown answers.

Figure 13: s(CASP) encoding of Example A1 (blinking light), based on Example 1 (light on/off).

```
1 happens(turn_light_off, T) :- holdsAt(light_on, T, 1).
2 happens(turn_light_on, T) :- not_holdsAt(light_on, T, 1).
```

(a) Solution (apx1/fix-holdsAt3.pl).

```
1 holdsAt(Fluent, T2, Dur) :- T2 #= T1 + Dur,
2   ..., happens(Event, T1), ...
```

(b) New axiom (axioms/bec_scasp.pl, line 80).

Figure 14: Solution for the problem in Example A1 (blinking light).

Inductive First-Order Formula Synthesis by ASP: A Case Study in Invariant Inference*

Ziyi Yang George Pîrlea Ilya Sergey
National University of Singapore, Singapore
yangziyi@u.nus.edu gpirlea@u.nus.edu ilya@nus.edu.sg

We present a framework for synthesising formulas in first-order logic (FOL) from examples, which unifies and advances state-of-the-art approaches for inference of transition system invariants. To do so, we study and categorise the existing methodologies, encoding techniques in their formula synthesis via answer set programming (ASP). Based on the derived categorisation, we propose *orthogonal slices*, a new technique for formula enumeration that partitions the search space into manageable chunks, enabling two approaches for incremental candidate pruning. Using a combination of existing techniques for first-order (FO) invariant synthesis and the orthogonal slices implemented in our framework FORCE, we significantly accelerate a state-of-the-art algorithm for distributed system invariant inference. We also show that our approach facilitates *composition* of different invariant inference frameworks, allowing for novel optimisations.

1 Introduction

First-Order Logic (FOL) has been used with great success as a foundational tool for modelling and verifying complex systems. Its applications span various domains, ranging from hardware design [5] to software verification [21]. These successes are largely attributed to the development of advanced frameworks that allow for automated verification and synthesis, often supported by high-performance provers such as Z3 [19] and cvc5 [1].

To achieve *fully automated* verification of a complex system in FOL, it is often necessary to *synthesise* formulas capturing the invariants (*i.e.*, the properties always hold) of the system being verified. Despite the undecidable nature of this task, many recent efforts have made substantial progress to infer inductive invariants of complex distributed systems (*e.g.*, Lamport's Paxos consensus protocol [16]) by synthesising FO formulas from examples: sampled traces of a protocol or counter-examples to induction. The resulting approaches are implemented by a plethora of distinct frameworks [27, 8], and a systematic study of their inter-connections is still missing. This raises important questions: are the existing synthesis methods fundamentally non-overlapping? Could techniques developed for one approach be adapted to benefit others? Addressing these questions would not only deepen our understanding of the underlying methodologies but also enable the development of superior tools for formula synthesis, potentially improving scalability of automated verification tools across various domains.

In this paper, we present a unified framework for synthesising bounded first-order formulas from examples—first-order structures that the formulas satisfy. Our framework is designed to encode and *combine* diverse synthesis techniques, enabling seamless integration with different high-level algorithms

*We thank Andrew Cropper, Roland Kaminski, and Jianan Yao for providing help on Popper, Clingo, and DuoAI, respectively. We are also grateful to Roland Yap for his feedback on a draft of this paper. This work was partially supported by a Singapore Ministry of Education (MoE) Tier 3 grant "Automated Program Repair" MOE-MOET32021-0001 and by Amazon Research Award "Scaling Automated Verification of Distributed Protocols with Specification Transformation and Synthesis".

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 511–527, doi:10.4204/EPTCS.439.35

© Z. Yang *et al.*
This work is licensed under the
Creative Commons Attribution License.

for invariant inference. To achieve this, we have conducted a detailed study of *nine* recent approaches for invariant inference of distributed systems (DS), each of which offered a different take on inductive synthesis of FO formulas. We summarise our study in two main observations *w.r.t.* inductive synthesis of FO formulas, hinting an opportunity for improvement in the state of the art:

O1 Existing techniques fall within a small number of distinct classes, in terms of how they treat their inputs and results, *e.g.*, how examples are used and how formulas are constructed. We give a uniform categorisation of these approaches.

O2 Whilst a wide variety of synthesis techniques exists, the vast majority fall into one of two categories, exploiting the properties of the first-order theories they employ: "redundancy elimination" and "incremental pruning".

To make a unified framework that captures **O1**, we use Answer Set Programming (ASP) [17], to encode the enumeration-based FO formula synthesis (as constraint solving) and the customisations and techniques of the synthesis (as knowledges representation). To improve on the existing techniques, we exploit **O2** by proposing the idea of *orthogonal slices* (also implemented by ASP) of the FO search space: a new approach to efficiently prune candidate formulas during the inductive synthesis. The key idea of orthogonal slices is to partition the search space into ordered slices, where the synthesis of former slices can be used to prune the latter slices using *either satisfied or unsatisfied* formulas. The practical benefits of the unified framework, FORCE (**F**irst-**O**rder synthesiser via o**R**thogonal sli**CE**s), are demonstrated by improving two state-of-the-art DS invariant synthesisers, DuoAI [27] and Flyvy [8], *without any conceptual modifications to their high-level algorithms*. Our results show that our framework is sufficiently expressive and extensible to encode and compose existing formula synthesis techniques, advancing the state of the art in DS invariant synthesis.

2 Overview

We start with a primer on inductive synthesis of FO formulas—a common subroutine in existing invariant inference frameworks for distributed systems (DS). Then we summarise nine notable existing approaches for DS invariant inference (with the earliest dated 2019), concluding with a brief description of our ASP-based framework to capture various aspects of the synthesis and its particular instance, orthogonal slices.

2.1 Problem Definition

First-Order Language. In this work, we focus on system properties that are expressible in a first-order, many-sorted logic with equality, following the common textbook definitions. A *signature* $\Sigma = \langle C, R, F, S \rangle$ consists of: a set of *constant symbols* C, a set of *relation symbols* (predicates) R, a set of *function symbols* F, and a set of *sorts* S for the variables, constants, and function symbols. In the rest of this paper we assume all signatures to be *finite*, *i.e.*, the sets C, R, F, and S are finite.

Logic *terms* are defined recursively. A term is either a *constant* $c \in C$, a *variable* x, or a *function symbol* $f \in F$ applied to other terms (*e.g.*, $f(x_1, x_2)$). Logic *atoms* are the basic formulas formed by applying relation symbols from R or equality to terms of appropriate sorts; and a *literal* is an atom or its negation (*i.e.*, $\neg p(x)$). *Formulas* are constructed by closing literals under *logical connectives* (*i.e.*, conjunction $\wedge$ and disjunction $\vee$) and *quantification* (universal $\forall$ and existential $\exists$). It is well-known that any FO formula can be transformed into an equivalent formula in *prenex normal form*, where all quantifiers are placed at the beginning. In this structure, the *prefix* includes the quantifiers, and the

matrix consists of the remaining Boolean components. For example, the formula $\forall x : s_1\, \exists y : s_2.\ p(x,y) \vee (\neg q(x) \wedge r(y))$ is in prenex normal form, with prefix $\forall x : s_1\, \exists y : s_2$ and matrix $p(x,y) \vee (\neg q(x) \wedge r(y))$.

Definition 1 (First-Order Synthesis Problem) *Given a set of formulas Ω_0 over signature Σ and a set of FO structures $\sigma = \{M_1, \ldots, M_k\}$, where each M_i is a model over Σ, find a set of formulas $\Phi = \{\phi_1, \ldots, \phi_n\} \subseteq \Omega_0$ s.t. $\forall \phi \in \Phi$:*

$$
\begin{aligned}
&1.\ \forall M \in \sigma.M \models \phi, && \textit{(satisfies all input FO structures)},\\
&2.\ \mathit{FreeVars}(\phi) = \emptyset, && \textit{(closed formula)},\\
&3.\ \exists M \notin \sigma.M \not\models \phi, && \textit{(non tautology)},\\
&4.\ \forall \phi' \in \Phi.\phi \neq \phi' \wedge \phi \not\models \phi', && \textit{(no formula entails another)}.
\end{aligned}
$$

In other words, the problem is to find in search space Ω_0 a conjuncted set of well-formed formulas Φ that satisfy all the given first-order structures in σ. Such a conjunction describes the "most precise" formula that satisfies all the given structures, because there is no satisfied formula which is entailed by any ϕ_i. The problem can be seen as an instance of the general specification synthesis problem [22] in the setting of positive-only learning [26]. An example of the problem is illustrated in App. A.

It is worth noting that we make several assumptions and simplifications to the problem definition above: (1) a disjunction of all input FO structures is always a valid (though overfit) solution, but in practice, meaningful formulas are to be found in a size-restricted search space; (2) search spaces with function symbols and constants can be easily handled by introducing new literals [28, §4], so we will avoid discussing them in detail; (3) we assume users want to synthesise prenex DNF formulas, as DS invariants are commonly expressed in this form; our techniques can be extended to other FO formulas.

2.2 Taxonomy of Existing Invariant Inference Algorithms

Existing invariant inference methods use different formula enumeration techniques and employ a variety of optimisations to effectively reduce the search space. We summarise nine representative existing approaches: I4 [18], FOL-IC3 [13], IC3PO [11], SWISS [12], DistAI [28], P-FOL-IC3 [14], DuoAI [27], Scimitar [25], and Flyvy [8] (in the order of their publication dates).

The categories are based on five detailed sub-aspects. The first two sub-aspects are "system-level", capturing the high-level design of a synthesis framework, while the remaining three are "algorithm-level": they define the pruning techniques applied to the brute-force enumeration of the search space.

2.2.1 System-Level Aspects

Inference mode. This aspect determines the way the overall inference procedure uses FO formula synthesis. In particular, *one-shot* synthesis [18, 28, 27] generates all satisfied formulas in the search space given fixed input examples (*e.g.*, sampled traces of a distributed protocol), while *multi-shot* synthesis [25, 8] generates a set of candidate formulas incrementally, based on examples obtained incrementally (*e.g.*, counter-examples of current invariants). Combined approaches [13, 11, 12, 14] use system traces or counter examples to guide the multi-shot synthesis, calling the synthesis procedure multiple times.

Historically, the majority of multi-shot synthesis algorithms can be seen as extensions of IC3 [2], while one-shot synthesis can be considered as extensions of Houdini [7]. Crucially, both modes share the underlying FO synthesis problem similar to the definition of our synthesis problem, which means our FORCE framework can be used to improve most existing approaches.

Language restriction. To make the synthesis problem tractable, it is frequently defined for a subclass of first-order logic. The most common one is the Effectively Propositional Logic (EPR [23, 6]) fragment, a subset of FOL in which formulas can be transformed into equivalent propositional formulas, allowing for provably decidable verification.

While EPR and its extensions are applied in most existing approaches with a theoretical decidability guarantee, in practice, synthesis approaches often impose additional other syntactic constraints. For example, the k-pseudo-DNF (proposed in P-FOL-IC3 [14], and further applied to [8]) is a syntax restriction for practical efficiency: it is based on the observation that the invariant formulas written in such form are smaller than standard DNF, reducing the search space that needs to be explored. Specifically, a k-pseudo-DNF formula has the matrix in the form of $c_1 \rightarrow (c_2 \vee \ldots \vee c_k)$, where c_i is a conjunction of literals. This is essentially a heuristic of the search, which makes sense because implications are commonly used to express invariants.

DistAI [28] proposed sub-templates (further applied to [27]) for efficiency, exploiting the following property of first-order logic with equality:

$$\forall X_1, X_2 : T.mat(X_1, X_2) \equiv \forall X : T.mat(X, X) \wedge \forall X_1 \neq X_2 : T.mat(X_1, X_2) \tag{0}$$

With such a property, formula synthesis can avoid enumerating formulas on the LHS of Eq. 0 because they are equivalent to the conjunction of the two RHS formulas. Therefore, the only enumerations among two sub-templates on the right are required to synthesis the satisfied formula on the left, resulting in an accelerated enumeration.

More than syntactic constraints above, specific tools [13, 12, 25, 8] also define their own syntactic customisations. However, upon close examination of existing invariant inference implementations, we find that syntactic constraints are not extensible in many tools. For example, extending k-pseudo-DNF with sub-templates would require modifying any algorithm that manipulates formulas in P-FOL-IC3.

2.2.2 Algorithm-Level Aspects

Redundancy elimination. The most common pruning to apply is redundancy elimination, which is used in many synthesis tasks. The idea is to simply eliminate the formulas that are equivalent to each other. Approximations as "pruning by symmetry" (to be detailed in Sec. 3) are often used, where the symmetry of formulas under quantification is identified to prune away redundant ones. More than the syntactic-based equivalence, the redundancy can also be introduced by semantics (*e.g.*, tautology and contradiction), where many case-by-case rules are used in DistAI [28] and DuoAI [27]. An example of formulas eliminated by this kind of redundancy is the one containing $p(X) \wedge \neg p(X)$ or $p(X) \vee \neg p(X)$ as sub-expressions. Another approach to redundancy is elimination *canonicalisation*, which is implemented by Flyvy [8]: it also uses symmetry breaking, but defines a partial order of formula sets (instead of the individual formulas) to eliminate redundancy.

A more complex but very effective pruning strategy of this category is proposed in DuoAI [27], where a large number of DNF formulas are shown to be redundant by their *decomposition*: $A \equiv B \wedge C$. Similar to Eq. 0, the formula A can be decomposed into a conjunction of smaller formulas, so the original formula can be pruned if all the smaller formulas are in the search space. However, the authors of Flyvy (*cf.* [8, Appendix D]) found that the decomposition in DuoAI is unsound in certain cases (*i.e.*, it leads to over-pruning), and proposed an amended version. We found it not easy to switch the DuoAI implementation to the amended version, as it uses the intermediate results of the original decomposition in the overall synthesis loop, which also makes it challenging to fairly compare the efficiency of the two methods.

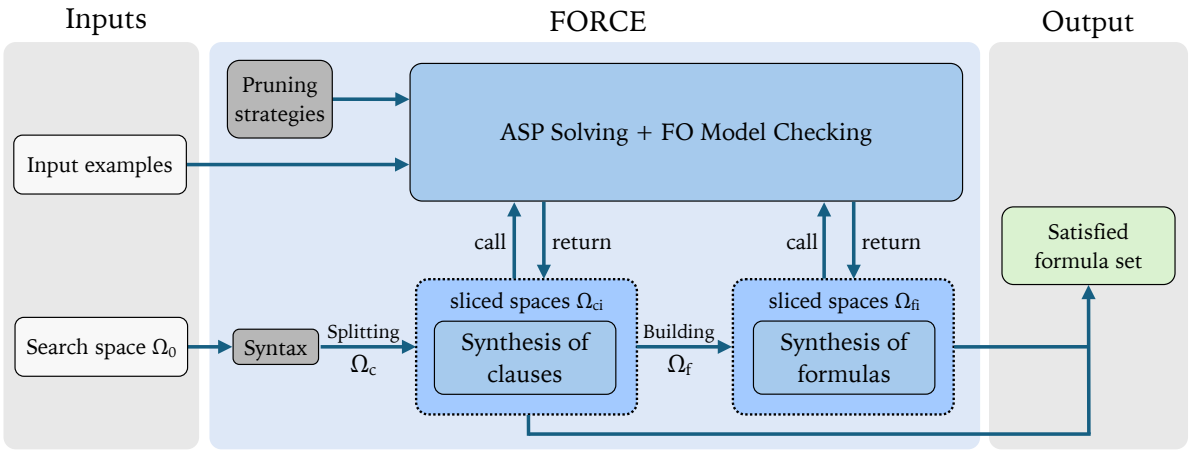


Fig. 1: The workflow of FORCE.

Incremental pruning. Another inherent part of nearly all efficient approaches to FOL formula synthesis is incremental pruning. The idea is to test formulas against the set of input FO structures in a specific order, exploiting the entailment relation between formulas to eliminate the need to test some of them.

The *implication graph* to be detailed in Sec. 4.1 is an effective technique of this aspect from DuoAI [27]. More than this, DuoAI also identify another incremental pruning technique is based on *co-implication*, which combines the intermediate results of FO model checking with the redundancy elimination to prune the search space. For example, if we find that formula $\forall x.P(x) \Rightarrow R(x)$ satisfies all examples, we do not need to the test any formulas of the form *prefix*. $(P(x) \wedge R(x) \wedge F) \vee G$, since they are co-implied by the original formula and *prefix*. $(P(x) \wedge F) \vee G$.

Other techniques. Several pruning techniques are introduced in existing invariant inference tools [13, 11, 25]. We will discuss them in Sec. 6, as they are not generally applicable to all FO synthesis tasks.

To sum up the taxonomy of existing approaches, it suggests a possibility of a unified framework to capture the existing techniques for FO synthesis and propose new ones. In particular, we find that the **Language** and **Redundancy** aspects together form a *static* search space, while the **Incrementality** aspect defines the *dynamic* pruning of the search space during the synthesis. These aspects are applicable to any FO synthesis problem.

2.3 FORCE: ASP-based Synthesis + Orthogonal Slices

Before the technical details, we give a high-level overview of our ASP-based synthesiser FORCE, whose workflow is shown in Fig. 1, by combining pruning strategies with the new *orthogonal slices* technique. The framework is customised by the grey parts: (1) the FO language (DNF by default) and (2) a set of pruning rules (predefined but extensible). Given the initial search space Ω_0 and a set of FO structures as input, FORCE starts by splitting the search space of clauses Ω_c from Ω_0. For further slices of Ω_c (called $\Omega_{c_1}, \Omega_{c_2}, \ldots$), the algorithm generates the formulas in Ω_{c_i} (by ASP solving), tests them on the examples (by model checking), and prunes the later slices ($\Omega_{c_j},\ j > i$) based on the results. Then using the output of Ω_c, the algorithm further builds the search space of formulas *other than clauses* Ω_f into $\Omega_{f_1}, \Omega_{f_2}, \ldots$, reusing the same process as in Ω_c to generate-test-prune. Finally, FORCE outputs both the clauses and

```
1: tmp ← init()
2: for all x ∈ Ω0 do
3:     if pruning1(x, tmp) then
4:         continue
5:     else if pruning2(x, tmp) then
6:         update2(tmp, x)
7:     else
8:         if Satisfied(x) then
9:             tmp.update(x)
   return tmp.result
```

Algorithm 1 Extending Def. 1 with Customised Pruning Rules

```
1: function FOSYN(Ω0, σ, prunings)
2:     solver ← init(Ω0, prunings)
3:     while solver.solve() do
4:         x ← solver.get_current()
5:         if Satisfied(x) then
6:             yield x
```

Fig. 2: The synthesis loop (imperative programming vs. ASP solving).

the non-clause formulas that satisfy the inputs. It is called *orthogonal* because (1) slicing Ω_0 into Ω_c and Ω_f and (2) slicing of Ω_c and Ω_f are using different pruning strategies and work together.

3 Static Search Spaces of First-Order Formulas in ASP

We assume the reader is familiar with the basics of ASP, such as rules, choices, aggregates, and refer to the literature for more details [9]. This section demonstrates how ASP is suitable for encoding the static search space of FO formulas.

Our high-level approach follows the "generate-and-test" workflow, similar to other invariant synthesis systems, but differs in how the search space and pruning techniques are encoded. ASP is a paradigm suitable for such exhaustive enumeration with restrictions (search space customisations and pruning techniques in our case). As shown in Fig. 2, existing imperative approaches (demonstrated on the left) require manual integration of pruning steps (line 3-6), which can lead to brittle and hard-to-maintain code due to dependencies on intermediate results and evaluation order. In contrast, our ASP-based approach on the right modularises the search space and pruning strategies directly into the solver (line 2). This ASP-based "generate-and-test" loop improves extensibility and maintainability, by allowing easier integration of new pruning rules (as new domain knowledge) and automatic handling of dependencies. In the remainder of this section, we first demonstrate how to encode a basic search for FO formulas and then show the search is customised with different knowledge of pruning.

3.1 Encoding the Enumeration

Let us show how to encode FO formula enumeration in ASP. As an illustration, we use DuoAI's [27] configuration of FO search space for synthesising invariants of the `lockserv` distributed protocol.

```
var: node: n1, n2; lock: l1
relations: lock_msg:          node, lock;  grant_msg:  node, lock;
           unlock_msg:        node, lock;  holds_lock: node, lock;
           server_holds_lock: lock
max-literal: 4 max-or: 3 max-and: 3 max-exists: 1
```

The first two lines, "var" and "relations", together specify the variables (in `node` and `lock` sorts) and atoms (made out of five relations and valid variables) that can be used in the formulas. Note that the order of variables in formulas' prefix is usually fixed in EPR formulas, *i.e.*, the variable `l1` cannot appear before any node variable (`n1`, `n2`) in the prefix. The search space of formulas in DNF is then constrained by the problem-specific customisations: the maximum number of literals in the formula, the maximum

number of disjunctions, the maximum number of cubes (*i.e.*, conjunctions of literals), and the maximum number of existential quantifiers. They together define the Ω_0 in Def. 1.

Thanks to the simplicity of FO formulas' prenex normal form, the enumeration can be easily encoded in ASP. The enumeration of formulas requires generating their two parts: prefixes and matrices, within the restrictions of the search space. The ASP encoding of the basic search space is illustrated as follows:

```
var(node, n1). var(node, n2). var(lock, l1).              % variables
0{exists(Var): var(Var, _)}1.                             % prefix bounded by max-exists
rel(lock_msg, (node, lock)). ...                          % relations
vars((node, lock), (n1, l1)). ...                         % variable tuples
atom(Pred, Args) :- rel(Pred, Types), vars(Types, Args). % atoms
pos(0..1). cube(1..3).                                    % sign of literal, max-or
0{lit_in_C(P,A,Pos,C):atom(P, A), pos(Pos)}3 :- cube(C). % matrix bounded by max-and
:- #count{P,A,Pos,C: lit_in_C(P,A,Pos,C)} >= 4.           % max-literal
```

In the program above, the prefix and matrix generation of formulas are achieved by the choice construct `0{...}n` on `exists()` and `lit_in_C()` predicates, which are restricted by the parameters in the configuration. The last line then eliminates the answer sets where more than four literals are in the corresponding formula. As an example, the answer set {`lit_in_C(lock_msg,(n1,l1),0,1)`, `lit_in_C(grant_msg,(n2,l1),1,1)`, `lit_in_C(unlock_msg,(n1,l1),0,2)`, `exists(l1)`} corresponds to $\forall$`n1 n2`,$\exists$`l1`. (¬`lock_msg(n1,l1)` ∧ `grant_msg(n2,l1)`) ∨ ¬`unlock_msg(n1,l1)`. We should also note that the encoding provided here is to illustrate the basic idea of the enumeration; for different syntactic customisations (*e.g.*, the variable order in the prefix) of FOL formulas, certain predicates and rules are required to be added to the encoding.

3.2 Encoding Pruning by Redundancy

A particular encoding of a search space can output many answer sets whose corresponding formulas not necessarily satisfy the examples or even basic well-formedness constraints. For instance, the formula $\forall n_1.\texttt{lock_msg}(n_2,l_1)$ should be ignored because n_2 is not in the prefix. As another example, $\mathit{prefix}.\texttt{lock_msg}(n_1,l_1) \vee \texttt{lock_msg}(n_2,l_1)$ is equivalent to $\mathit{prefix}.\texttt{lock_msg}(n_2,l_1) \vee \texttt{lock_msg}(n_1,l_1)$ but can be featured twice as two different answer sets. Our next step is, therefore, to encode those **Redundancy** techniques from Sec. 2.2 on the search space to only output well-formed formulas.

Let us illustrate the *symmetry-based normalisation* technique allowing to exploit equivalence using ASP; the remaining encoding of redundancy elimination can be found in our implementation. The normalisations are done by building a partial order on the formulas. That is, we can make sure that (1) the number of literals in cubes (from left to right in a DNF) is non-decreasing, (2) the minimal (in alphabetic order) literal in a cube i is less than the minimal literal in cube $j > i$ if their number of literals are the same, and (3) if two variables of one sort $v_i < v_j$, then the minimum predicate where v_i appears should be less or equal to the minimum predicate where v_j appears. The following ASP program encodes these pruning rules by constraining the orders in the formula.

```
lit_no_in(No, C) :- lit_in_C(P,A,Pos,C), lit_no(P,A,Pos,No).
num_lit(C, N) :- cube(C), #count{P,A,Pos: lit_in_C(P,A,Pos,C)} = N.
:- num_lit(C1,N1), num_lit(C2,N2), C1 < C2, N1 > N2.
min_lit(C, Min) :- cube(C), #min{No:lit_no_in(No,C)} = Min.
:- num_lit(C1,N), num_lit(C2,N), min_lit(C1,Min1), min_lit(C2,Min2), C1<C2, Min1>Min2.
min_lit_var(V, Min) :- used_var(V), #min{P:lit_in_C(P,A,_,_), var_in(V,A)} = Min.
:- min_lit_var(V1,Min1), min_lit_var(V2,Min2), V1 < V2, Min1 > Min2.
```

$$\forall x.P(x) \vee R(x) \quad (1)$$
$$\forall x.P(x) \vee S(x) \quad (2)$$
$$\forall x.P(x) \vee (R(x) \wedge S(x)) \quad (3)$$

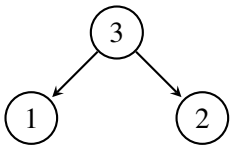


Fig. 3: Left: Formulas example. Right: Implication graph of the formulas.

To summarise this section, we have shown that for the synthesis of FO formulas, the existing techniques can be easily encoded in ASP, which works as the backbone of our synthesiser. The expressive power of ASP allows us to encode existing techniques shown in Sec. 2.2 concisely and make them work together efficiently, being further optimised with our new pruning technique, described in the next section.

4 Dynamic Search Spaces via Orthogonal Slices

Armed with the ASP-based framework formula enumeration in the previous section, which allows one to combine pruning techniques, we propose *orthogonal slices* to handle the dynamic search space, which (1) generalises the state-of-the-art incremental pruning technique *implication graph* (IG), and (2) introduces a novel complementary pruning to resolve the bottleneck of IG. Both are achieved by slicing the FO search space into smaller ordered parts, and easily implemented by ASP's incremental solving.

4.1 Implication Graphs

Amongst the proposed pruning techniques for distributed system invariant synthesis in Sec. 2.2, *implication graph* of FO formulas is a prominent one. An IG is a directed graph where each node represents a formula, and an edge from A to B indicates that A implies B. The pruning is processed by removing formulas that are implied by already satisfied formulas from the search space, dynamically, to accelerate the synthesis process. That said, we can call it "pruning by satisfied formulas". The IG-based tool DuoAI [27] can infer inductive invariants for many complex protocols where no other existing tools can.

Example 1 (An Illustration of the Implication Graph Shown in Fig. 3) *Formula* (3) *implies both formula* (1) *and formula* (2)*. If formula* (3) *is satisfied, then formulas* (1) *and* (2) *are automatically satisfied and can be pruned from the search space.*

4.2 From the Formula to the Search Space

The first step of the orthogonal slices is to *abstract* the entailment relation from formulas to the level of the search spaces, then synthesise formulas following the partial order on the search spaces to achieve incremental pruning. We represent a FO formula search space by means of imposing syntactic constraints of the candidates, with the sets of possible values as arguments. As such, the search space is the Cartesian product of those sets, where each element describes a sub search space.

Definition 2 (Template of First-Order Formulas and Its Slicing) *A template T of first-order formulas is defined by a tuple of parameter sets $T = (P_1, \ldots, P_n)$, where each P_i represents a set of possible values for a parameter. The search space $\Omega(T)$ defined by T is the Cartesian product of these parameter sets. A valid slicing of T (called $\mathtt{SL}(T)$) is defined as a partitioning of T into sliced-templates $\{T_1, \ldots, T_j\}$,*

where each sliced-template $T_i = (P_{i1}, \ldots, P_{in})$ corresponds to a subset of the parameter sets. The search space $\Omega(T)$ is then partitioned into subsets $\{\Omega(T_1), \ldots, \Omega(T_j)\}$, where:

$$\Omega(T) = \prod_{i=1}^{n} P_i. \qquad \bigcup_{i=1}^{j} \Omega(T_i) = \Omega(T), \quad \textit{and} \quad \Omega_m \cap \Omega_n = \emptyset \quad \textit{for all } m \neq n.$$

This ensures that the entire search space is covered, without overlap between slices, and each slice Ω_i corresponds to a sliced-template T_i.

Example 2 (A Template of `lockserv` in Sec. 3.1) • *The number of existential quantifiers ne,*

- *The number of variables used in each sort in a tuple $tv = (n_{v1}, \ldots, n_{vi}, \ldots)$,*
- *The number of literals in each cube sorted in a tuple $tl = (n_{l1}, \ldots, n_{li}, \ldots)$.*

And the parameters of the template (after the redundancy pruning in Sec. 3.2) are:

- $P_{ne} = \{0, 1\}$
- $P_{tv} = \{(0,1), (1,0), (1,1), (2,0), (2,1)\}$
- $P_{tl} = \{(1), (2), (3), (1,1), (1,2), (1,3), (2,2), (1,1,2)\}$

Now we describe the slicing for "pruning by satisfied formulas", which is defined by the partial order of the sliced-templates (and their parameters).

Definition 3 (Partial Order for Parameters of Templates) *Given two parameters P_{ix} and P_{iy} as subsets of P_i in a template $T = (P_1, \ldots, P_i, \ldots, P_n)$, we say that P_{ix} is less or equal than P_{iy} (denoted as $P_{ix} \preceq P_{iy}$; $P_{ix} \prec P_{iy}$ in case $P_{ix} \neq P_{iy}$)* w.r.t. $\mathtt{SL}(T)$ *if*

$$\forall T_1 = (P'_1, \ldots, P_{ix}, \ldots, P'_n),\ T_2 = (P'_1, \ldots, P_{iy}, \ldots, P'_n) \in \mathtt{SL}(T),$$
$$\forall \phi \in \Omega(T_1),\ \exists \phi' \in \Omega(T_2) \textit{ such that } \phi' \models \phi.$$

Definition 4 (Partial Order for Sliced-templates) *Given two sliced-templates $T_i = (P_{i1}, \ldots, P_{in})$ and $T_j = (P_{j1}, \ldots, P_{jn})$ sliced from $\mathtt{SL}(T)$, we say that T_i is less or equal than T_j (denoted as $T_i \preceq T_j$) if $\forall k \in [1,n], P_{ik} \preceq P_{jk}$, where the equality holds if and only if $P_{ik} = P_{jk}$ for all $k \in [1,n]$.*

With two definitions above, the whole search space of FO formulas is sliced (and ordered) by the partial order; we call this procedure SplitTem. Note that possibly the search space's parameters are not as regular as in Ex. 2, but the worst case of the partial order is exactly the implication graph: the set of possible formula candidates is the only parameter, and the partial order is defined by FO entailment. This says, the parameterisation of FO search space by syntactic constraints is "always" possible.

Example 3 (Partial Order of the Templates in Ex. 2) *The partial order of the templates' parameters of `lockserv` is defined as follows:*

- *For n_e, it follows the integer order: $\{0\} \prec \{1\}$.*
- *For t_v, the set $\{(n_{v1}, n_{v2}, \ldots, n_{vi}, \ldots)\} \prec \{(n_{v1}, n_{v2}, \ldots, n_{vi}+1, \ldots)\}$*
- *For t_l, the set $\{(n_{l1}, n_{l2}, \ldots, n_{li}, \ldots)\} \prec \{(n_{l1}, n_{l2}, \ldots, n_{li}-1, \ldots)\}$ and $\{(n_{l1}, n_{l2}, \ldots, n_{li}, \ldots)\} \prec \{(n_{l1}+1, n_{l2}, \ldots, n_{li}, 1)\}$*

The partial order on template parameters is defined consistently with formula entailment relations: by swapping $\forall$ into $\exists$, a formula becomes more general; by replacing one variable with a fresh one (together with the formula decomposition, proven in [28, §4]), a formula becomes more general; by deleting a literal from a cube or adding a new cube, a formula becomes more general. Therefore, by obtaining the partial order of a slicing of the search space, the "pruning by satisfied formulas" is naturally achieved.

The ASP encoding of "pruning by satisfied formulas" is done by multi-shot solving [10], which is standard to achieve incremental solving in ASP. The sketch of its encoding is as follows:

```
#program inv(prefix,matrix).
:- output(Prefix, Matrix), pre_weaken(Prefix, prefix), mat_weaken(Matrix, matrix).
```

which says that the formula entailed by the `inv` (*i.e.*, a satisfied formula) should not be generated. The entailment checking is implemented by variable substitution together with prefix and matrix weakening (detailed in our implementation).

4.3 Slicing DNF Modulo Clauses

The problem of implication graph (or "pruning by satisfied formulas" in general) is its scalability: the search starts from the root to leaves of the graph (top to bottom in Fig. 3), but the number of root nodes is still exponential in the search space's size. To see the issue, let us take the formula *prefix*. $(lit_{11} \wedge lit_{12} \wedge lit_{13}) \vee (lit_{21} \wedge lit_{22})$ as an example: it can be a root node used to prune the formula *prefix*. $lit_{1i} \vee lit_{2j}$ because of the entailment. However, checking all root formulas has complexity of $O(n^5)$ (n is the number of literals), but this effort prunes only formulas using $O(n^2)$ space.

Intuitively, if "pruning by satisfied formulas" is costly when using the result of a larger search space to prune the smaller one, its dual version–"pruning by unsatisfied formulas", should solve it. This idea, however, is not immediately applicable for two reasons: (1) the two pruning strategies have different directions (general-to-specific v. specific-to-general), which means an algorithm needs to deal with both, and (2) for complex problems, the majority of formulas are unsatisfied (they do not cover all examples), which means reducing the search space too many times incurs a large performance overhead.

Our solution to the first issue is simply another slicing, which is orthogonal to the slicing of templates: first synthesising clauses (disjunctions of literals) by slicing them from the whole search space (named SPLITDNF), and then pruning the search of DNF synthesis based on the unsatisfied clauses. To further address the second issue, we avoid the high-cost of "pruning by unsatisfied clauses" by *constructing* the DNF search space from the satisfied clauses. The definition of DNF search space construction given below describes the pruning of this slicing.

Definition 5 (Possibly satisfied DNF modulo clauses) *Given a set of satisfied (*w.r.t. *input examples) clauses Φ_c, a formula in DNF of the form prefix.* $(lit_{11} \wedge \ldots \wedge lit_{1k_1}) \vee \ldots \vee (lit_{m1} \wedge \ldots \wedge lit_{mk_m})$ *is possibly satisfied only if*

$$\forall l_i \in \bigcup_{i=1}^{m} \{lit_{i1}, \ldots, lit_{ik_i}\},\ \textit{prefix}.\ l_1 \wedge \ldots \wedge l_m$$

is satisfied w.r.t. Φ_c. *The function constructing DNFs is denoted as* BUILDFROMCLAUSES.

In plain words, a formula in DNF is added into the search space only when all clauses it entails satisfy all input FO structures. The reason we resolve the bottleneck of "pruning by satisfied formulas" is evident: the search space of clauses, which is exponential in the number of cubes, is much smaller than the search space of DNF (exponential in the number of literals) for most cases. Moreover, since this search space construction is also naturally encoded by ASP, all existing techniques to formulas in Sec. 2.2 are directly applied to further refine this search space built from clauses.

Synthesis	ASP	Connection
Parameterised search space	choice constructs	set Cartesian product
Slicing	external statement	set partition
Static pruning	integrity constraint	set comprehension
Dynamic pruning	multi-shot solving	set union, difference

Tab. 1: The connection between synthesis and ASP.

4.4 The Synthesis Algorithm

Given the formula synthesis algorithm in Fig. 1 for an input search space, and the two approaches to slice the search space, the procedure for synthesising formulas is given by Algorithm 2: first synthesise clauses by its sliced template (lines 4 to 7), then synthesise other formulas by the sliced template of DNF modulo clauses (lines 9 to 12), finally normalising the satisfied formulas and output (lines 13 and 14). The input *pruning* rules are customisable, but we pre-defined existing redundant and incremental rules in Sec. 2.2. The soundness of the algorithm *w.r.t.* orthogonal slices (*i.e.*, it does not over-prune) is guaranteed by the fact that if a formula A is pruned, A is either unsatisfied, having being pruned by unsatisfied clauses in DNF modulo clauses, or satisfied but more general than a satisfied formula which prunes A in a sliced template.

Algorithm 2 The Core Algorithm of FORCE

```
1:  function FORCE(Ω0, σ, prunings = pre_def)
2:      Φc, Φf ← ∅
3:      Ωc ← SPLITDNF(Ω0)
4:      for Ωci ∈ SPLITTEM(Ωc) do
5:          Φi ← FOSYN(Ωci, σ, prunings)
6:          Φc ← Φc ∪ Φi
7:          prunings ← prunings.update(Φi)
8:      Ωf ← BUILDFROMCLAUSES(Φc, Ω0)
9:      for Ωfi ∈ SPLITTEM(Ωf) do
10:         Φi ← FOSYN(Ωfi, σ, prunings)
11:         Φf ← Φf ∪ Φi
12:         prunings ← prunings.update(Φi)
13:     Φc ← FILTERIMPLIED(Φc, Φf)
14:     return Φc ∪ Φf
```

Looking back to the whole synthesis process, both static and dynamic search spaces we discussed are essentially achieved by different set operations, which is the algorithmic reason for us to use ASP. In Tab. 1, we summarise the connection between the synthesis problem and ASP by unifying them as set operations, which hopefully helps the readers from either synthesis or ASP background to understand "why and what synthesis task is suitable for ASP".

5 Experimental Evaluation

5.1 Implementation and Settings

Our implementation of FORCE combines the use of ASP (250 lines for the enumeration plus 80 lines for pruning, which is extremely concise to encode the different existing and new pruning techniques), and C++ to call APIs of Clingo [9] (400 lines for the synthesiser call functions, and 200 lines to integrate with other systems). The benchmarks are run on a MacBook Pro with 8-core M1 Pro CPU, 16GB RAM. The current version of FORCE is available at `https://zenodo.org/records/15654427`.

We ran FORCE on a set of benchmarks from the work on DuoAI [27], the most efficient state-of-the-art DS invariant inference tool. Briefly speaking, DuoAI's algorithm is an enumeration-based inductive synthesis using sampled traces of the protocols as examples, followed by an optimised Houdini [7]. Since the formula enumeration component has non-negligible run time (detailed later), reducing the overhead

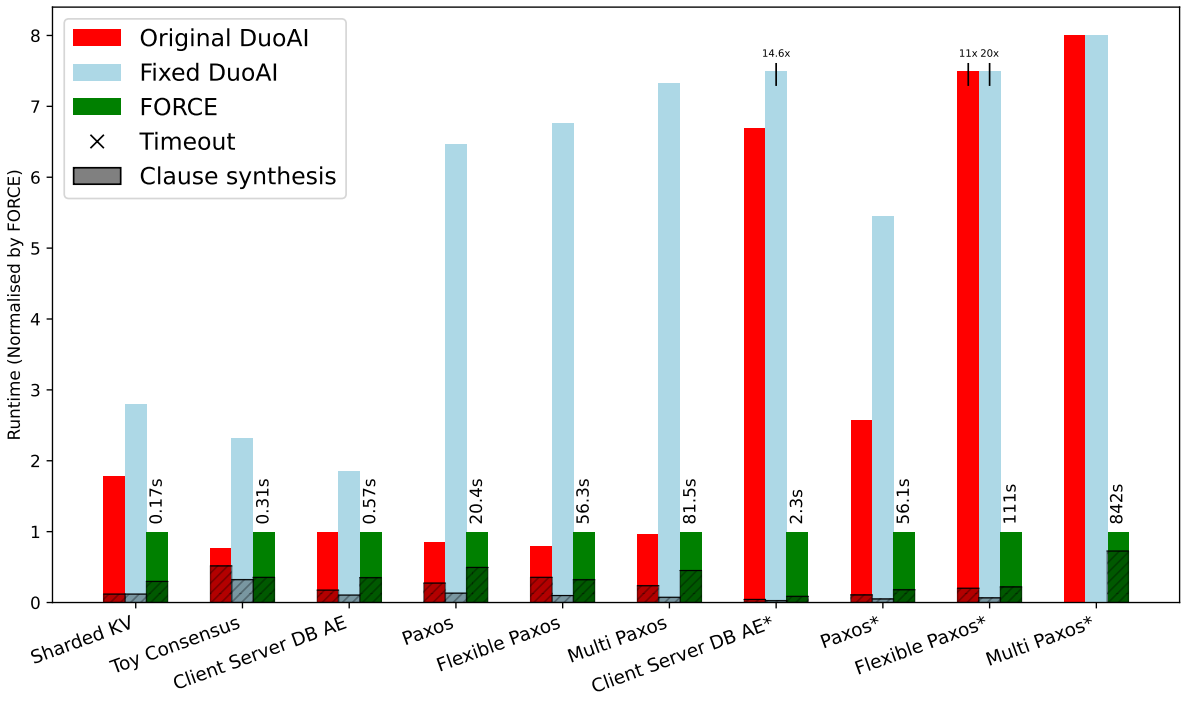


Fig. 4: Invariant synthesis. Results are normalised *w.r.t.* FORCE run times.

of the enumeration part can significantly improve the overall performance.

Note that (at least) two unsound optimisations were made in DuoAI: the first is an *over-pruning*, found and rectified by [8]: DuoAI identifies formulas that can be decomposed into smaller formulas, then not testing the original formula to reduce the search space; however, a subset of those should not be decomposable, which leads to unsoundness by missing possibly satisfied formulas. The second unsound optimisation is the use of *restricted quantifiers* of formulas; specifically, the default setting of DuoAI only allows arbitrary quantifiers over the last three variables in a template (*e.g.*, for $Q_1X_1.Q_2X_2.Q_3X_3.Q_4X_4$, Q_1 can only be $\forall$), which results in an incomplete search in general, but practically works for their benchmarks. We treat the first issue as a bug, and the second observation as domain-specific knowledge that encodes an extra constraint on the search space.

5.2 Results and Analysis

Our performance statistics are shown in Fig. 4. We selected six complex distributed protocols from the DuoAI suite for evaluation on synthesising inductive invariants. FORCE was given the same input as DuoAI: traces as input examples and a search space configuration which contains the inductive invariant. For each protocol, we benchmark with two configurations: one with the restricted quantifier limitation from DuoAI and one without it. The unrestricted quantifier setting is indicated by a $*$ in the figure when enabled. The reason for showing only 10 data points, rather than 12, is that the difference of quantifier restriction did not affect the small search space for the first two smaller protocols. We fixed the over-pruning bug in DuoAI (referring to the result as "fixed DuoAI") with our best effort for a fair comparison. That said, we also provide results of the "original" DuoAI version as a reference point.

The results show that FORCE significantly outperforms fixed DuoAI in all benchmarks, and also

beats the original (unsound) DuoAI in most cases. The difference in performance is larger without the quantifier restriction. As a reference, the original DuoAI's runtime to synthesise invariants for Paxos, FlexiblePaxos and MultiPaxos are 60.4s, 78.7s and 1,549s, respectively; this means the enumeration in DuoAI is the main bottleneck for complex protocols, so the improvement is effective for overall runtime.

The improvement is mainly sourced from the effective pruning by DNF modulo clauses (Sec. 4.3): as the *clause synthesis* part in the bars shows, FORCE takes (slightly) longer time on the clause synthesis than DuoAI, and results in a much shorter time for the remaining synthesis by reducing the search space. The huge difference for cases without quantifier restriction is also explained for the same reason: taking Paxos as an example, the number of satisfied clauses increased from 134 to 141 when disabling the restriction in FORCE, which means the increased search space for DNF is not much (with 36s difference of time); but for DuoAI, the enumeration explores the whole new search space (thus 76s difference in total). Note that abstracting the formula into the search space as in Sec. 4.2 allows parallelism of the formula synthesis in FORCE, which also contributes to the efficiency, but it is specific to (1) the implementation of parallelism, and (2) the multi-threaded solving in Clingo, which is not essential to discuss here. The overall extra time when disabling the parallelism in our system varies from 50% to 100%, which shows potential improvement with a better implementation of parallelism.

Another interesting comparison *w.r.t.* to the unsoundness in DuoAI is shown between the red and blue bars: the fixed version takes a longer time to explore the (now larger) search space, but the clause synthesis time is lower, because additional discovered satisfied formulas prune the clauses entailed (which also implies the unsoundness of original DuoAI). By analogy with our orthogonal slices, they are using results from "larger slices" to prune the "smaller slices", which results in inefficiency, as expected (see Sec. 4.3). From a high-level aspect, this comparison illustrates the common trade-off between completeness and efficiency in synthesis tasks, but FORCE achieves complete search with even better efficiency comparing to the original DuoAI by spending minor extra time (in the green bars) to synthesise clauses.

5.3 On Composability of FORCE

More than the performance benefits of the orthogonal slices, the "solver-aided" feature of FORCE is also promising: it works as a general framework to combine different approaches by providing a common interface for FO synthesis. To illustrate this, we built the bridges for both DuoAI and Flyvy [8] (about 200 lines of code for each), and used the clause output (*i.e.*, part of Fig. 2 before line 8) to optimise Flyvy.

The high-level task of Flyvy is to output the *strongest* inductive invariant (that contains a set of FOL formulas) given a distributed system protocol and a bounded FO language. Without describing the details of Flyvy's algorithm, it is clear that if Flyvy is obtaining formulas, those formulas can be bounded by DNF modulo clauses. We use the traces obtained from DuoAI together with the bounded language to synthesise satisfied clauses and output them as an ASP program by FORCE, then restricting the formulas in Flyvy by calling the ASP program to filter out the unnecessary formulas. The results show that, for two most complex protocols can be synthesised in Flyvy, Paxos and FlexiblePaxos, the sizes of the output invariants are reduced by about 13% (165/1260) and 7% (61/816) respectively on average (detailed in App. B), and they remain strongest in the sense that unnecessary formulas are removed during the intermediate steps of Flyvy. We believe it is a promising direction to explore different combinations of existing approaches with the power of FORCE.

6 Related Work and Discussion

Inductive Logic Programming. Our work is closely related to the field of Inductive Logic Programming (ILP, [20, 3]), which focuses on learning *logic programs* inductively. Concretely, the state-of-the-art

ILP system Popper [4] shares a similar workflow with FORCE: in a "generate-test-prune" loop, both systems use ASP to generate the candidate and prune the search. There are two main differences between FORCE and Popper: (1) the preferred candidate (among all correct solutions) in ILP is the most general program because both positive and negative examples are given; while in FORCE, the output is the conjunction of most specific formulas that satisfies all positive examples, because "logical true" is always a valid general formula in absence of negative examples. (2) the properties of FO formulas expose more domain knowledge to be encoded by ASP for pruning compared to Horn clauses.

Other Approaches for Invariant Inference. Among the techniques shown in Sec. 2.3, techniques from three tools (in the **Other techniques** category) have not been encoded into FORCE yet. Two of them share similarities in how they improve the inference process: FOL-IC3 [13] is an inference framework building on a first-order logic *separability* solver; IC3PO [11] is an invariant inference tool based on the relation between the *symmetry* of satisfied FO structures and the necessary *quantification* in satisfying formulas in FOL. Both of them are based on PDR/IC3 [2], where the generalisation of FO structures is performed based on a small set of examples (usually less than 10, in contrast with DuoAI, SWISS, and FORCE, where the generalisation is among thousands of examples). From a technical perspective, their techniques can be encoded into FORCE, since both are SAT-based. However, the scalability of their techniques make them not generally applicable to large sets of examples, which we will discuss later.

Scimitar [25], unlike most of the related works where a "global" inductive invariant is inferred, is a tool that infers "local" invariants for each transition state in the distributed system protocol. It builds an *inductive proof graph* that abstracts the protocols, and locally synthesises the invariants for each node in the graph, where the local synthesis problem is an instance of our FO synthesis.

Discussion on Inductive Generalisation. The least general generalisation [24] is known to be a foundational result in the field of bottom-up inductive logic programming, where the given programs (bottoms) are generalised to the general program (top). In contrast, FORCE is a top-down approach by "generate-and-test". Notably, in the domain of distributed systems invariant inference, the authors of the SWISS tool [12] described their "failed attempt" to use constraint solving for bottom-up generalisation (without a detailed explanation). This is coincident with our observation that the bottom-up generalisation (like FOL-IC3 and IC3PO) is not scalable to large sets of examples compared to the top-down approach. We provide our own explanation: given a whole search space, any example can be regarded as a constraint to prune the search; while the cost for the constraints can be considered linear in the number of examples, the benefit of pruning decreases as the number of constraints increases. This brings a possible further work to combine the top-down and bottom-up approaches by using subsets of examples for utilising the bottom-up generalisation.

Outside the domain discussed above, [15] is a notable work from automata theory that studies the learnability (*w.r.t.* generalisation) of formulas in finite variable logic. It does not focus on the concrete algorithm in practice, but show potential to extend FORCE to logics more general than FOL.

7 Conclusion

In this work, we proposed a unified ASP-based framework FORCE for synthesising formulas in first-order logic from examples. To do so, we used ASP as a framework for implementing inductive formula synthesis offering constraint solving to encode the search and rule-based knowledge to prune the search space. Using our ASP-based encoding, we proposed *orthogonal slices*—a novel technique that significantly accelerates formula synthesis. Finally, we have shown that declaratively capturing the essence of different approaches for formula synthesis in ASP enables a more efficient and composable solution.

References

[1] Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli & Yoni Zohar (2022): *cvc5: A Versatile and Industrial-Strength SMT Solver*. In: *TACAS*, *LNCS* 13243, Springer, pp. 415–442, doi:10.1007/978-3-030-99524-9_24.

[2] Aaron R. Bradley (2011): *SAT-Based Model Checking without Unrolling*. In: *VMCAI*, *LNCS* 6538, Springer, pp. 70–87, doi:10.1007/978-3-642-18275-4_7.

[3] Andrew Cropper & Sebastijan Dumancic (2022): *Inductive Logic Programming At 30: A New Introduction*. *J. Artif. Intell. Res.* 74, pp. 765–850, doi:10.1613/JAIR.1.13507.

[4] Andrew Cropper & Rolf Morel (2021): *Learning programs by learning from failures*. *Machine Learning* 110(4), pp. 801–856, doi:10.1007/S10994-020-05934-Z.

[5] Niklas Eén, Alan Mishchenko & Robert K. Brayton (2011): *Efficient implementation of property directed reachability*. In: *FMCAD*, FMCAD Inc., pp. 125–134.

[6] Moshe Emmer, Zurab Khasidashvili, Konstantin Korovin & Andrei Voronkov (2010): *Encoding industrial hardware verification problems into effectively propositional logic*. In: *FMCAD*, IEEE, pp. 137–144.

[7] Cormac Flanagan & K. Rustan M. Leino (2001): *Houdini, an Annotation Assistant for ESC/Java*. In: *FME*, *LNCS* 2021, Springer, pp. 500–517, doi:10.1007/3-540-45251-6_29.

[8] Eden Frenkel, Tej Chajed, Oded Padon & Sharon Shoham (2024): *Efficient Implementation of an Abstract Domain of Quantified First-Order Formulas*. *CoRR* abs/2405.10308, doi:10.48550/ARXIV.2405.10308. arXiv:2405.10308.

[9] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2012): *Answer Set Solving in Practice*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, doi:10.2200/S00457ED1V01Y201211AIM019.

[10] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2019): *Multi-shot ASP solving with clingo*. *Theory Pract. Log. Program.* 19(1), pp. 27–82, doi:10.1017/S1471068418000054.

[11] Aman Goel & Karem A. Sakallah (2021): *On Symmetry and Quantification: A New Approach to Verify Distributed Protocols*. In: *NASA Formal Methods*, *LNCS* 12673, Springer, doi:10.1007/978-3-030-76384-8_9.

[12] Travis Hance, Marijn Heule, Ruben Martins & Bryan Parno (2021): *Finding Invariants of Distributed Systems: It's a Small (Enough) World After All*. In: *NSDI*, USENIX Association, pp. 115–131. Available at `https://www.usenix.org/conference/nsdi21/presentation/hance`.

[13] Jason R. Koenig, Oded Padon, Neil Immerman & Alex Aiken (2020): *First-order quantified separators*. In: *PLDI*, ACM, pp. 703–717, doi:10.1145/3385412.3386018.

[14] Jason R. Koenig, Oded Padon, Sharon Shoham & Alex Aiken (2022): *Inferring Invariants with Quantifier Alternations: Taming the Search Space Explosion*. In: *TACAS*, *LNCS* 13243, Springer, pp. 338–356, doi:10.1007/978-3-030-99524-9_18.

[15] Paul Krogmeier & P. Madhusudan (2022): *Learning formulas in finite variable logics*. *Proc. ACM Program. Lang.* 6(POPL), pp. 1–28, doi:10.1145/3498671.

[16] Leslie Lamport (1998): *The Part-Time Parliament*. *ACM Trans. Comput. Syst.* 16(2), pp. 133–169, doi:10.1145/279227.279229.

[17] Vladimir Lifschitz (2002): *Answer set programming and plan generation*. *Artificial Intelligence* 138(1-2), pp. 39–54, doi:10.1016/S0004-3702(02)00186-8.

[18] Haojun Ma, Aman Goel, Jean-Baptiste Jeannin, Manos Kapritsos, Baris Kasikci & Karem A. Sakallah (2019): *I4: incremental inference of inductive invariants for verification of distributed protocols*. In: *SOSP*, ACM, pp. 370–384, doi:10.1145/3341301.3359651.

[19] Leonardo Mendonça de Moura & Nikolaj S. Bjørner (2008): *Z3: An Efficient SMT Solver*. In: *TACAS*, *LNCS* 4963, Springer, pp. 337–340, doi:10.1007/978-3-540-78800-3_24.

[20] Stephen H. Muggleton & Luc De Raedt (1994): *Inductive Logic Programming: Theory and Methods*. *J. Log. Program.* 19/20, pp. 629–679, doi:10.1016/0743-1066(94)90035-3.

[21] Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv & Sharon Shoham (2016): *Ivy: Safety Verification by Interactive Generalization*. In: *PLDI*, ACM, pp. 614–630, doi:10.1145/2908080.2908118.

[22] Kanghee Park, Loris D'Antoni & Thomas W. Reps (2023): *Synthesizing Specifications*. *Proc. ACM Program. Lang.* 7(OOPSLA2), pp. 1787–1816, doi:10.1145/3622861.

[23] Ruzica Piskac, Leonardo Mendonça de Moura & Nikolaj S. Bjørner (2010): *Deciding Effectively Propositional Logic Using DPLL and Substitution Sets*. *J. Autom. Reason.* 44(4), pp. 401–424, doi:10.1007/S10817-009-9161-6.

[24] Gordon D Plotkin (1970): *A note on inductive generalization*. *Machine intelligence* 5(1), pp. 153–163.

[25] William Schultz, Edward Ashton, Heidi Howard & Stavros Tripakis (2024): *Scalable, Interpretable Distributed Protocol Verification by Inductive Proof Slicing*. *CoRR* abs/2404.18048, doi:10.48550/ARXIV.2404.18048. arXiv:2404.18048.

[26] Ziyi Yang & Ilya Sergey (2025): *Inductive Synthesis of Inductive Heap Predicates*. *Proc. ACM Program. Lang.* 9(OOPSLA1), pp. 169–195, doi:10.1145/3720420.

[27] Jianan Yao, Runzhou Tao, Ronghui Gu & Jason Nieh (2022): *DuoAI: Fast, Automated Inference of Inductive Invariants for Verifying Distributed Protocols*. In: *OSDI*, USENIX Association, pp. 485–501. Available at https://www.usenix.org/conference/osdi22/presentation/yao.

[28] Jianan Yao, Runzhou Tao, Ronghui Gu, Jason Nieh, Suman Jana & Gabriel Ryan (2021): *DistAI: Data-Driven Automated Invariant Learning for Distributed Protocols*. In: *OSDI*, USENIX Association, pp. 405–421. Available at https://www.usenix.org/conference/osdi21/presentation/yao.

A An Example of Def. 1 in Sec. 2.1

This section provides an example of the FO synthesis problem.

Example 4 (A tiny example) *Considering a FO signature* $\Sigma = \langle C, R, F, S\rangle$ *with* C *and* F *empty,* $R = \{p, q, r\}$*, and* $S = \{X\}$*. The synthesis problem is defined with the inputs:*

- *the set of FO formulas is restricted to the form of* $\forall X : lit_1$ *or* $\forall X : lit_1 \vee lit_2$*, which is a disjunction of one or two literals with one universal quantifier. Without the pruning,* Ω_0 *contains* $(2*3)+(2*3)^2 = 42$ *formulas in total.*
- *the set of FO structures is* $\sigma = \{M_1, M_2\}$*, where* M_1 *and* M_2 *are two models over* Σ *sharing the universe* $\{x_0, x_1, x_2\} \in X$ *and the interpretations of p, q, and r are:*
  1. M_1*: [*$p^{M_1} = \{x_0,\ x_1\}$, $q^{M_1} = \{x_1,\ x_2\}$, $r^{M_1} = \emptyset$*]*
  2. M_2*: [*$p^{M_2} = \{x_0,\ x_1\}$, $q^{M_2} = \{x_2\}$, $r^{M_2} = \{x_1\}$*]*

And the output of the synthesis problem based on Def. 1 is a set of formulas

$$\Phi = \{\forall X.p(X) \vee q(X), \forall X.p(X) \vee \neg r(X)\}.$$

As a simple case of the sliced-template, our algorithm will first find satisfied formula in $\forall X : lit_1$ (and fails), then checking the satisfied formula in $\forall X : lit_1 \vee lit_2$ (with the two formulas successfully found). The inputs of FORCE, the bounded FO search space and the real traces of distributed system protocols, are much more complex. An interested reader can refer to `configs/` and `traces/` folders in https://github.com/verse-lab/FORCE for the real instances.

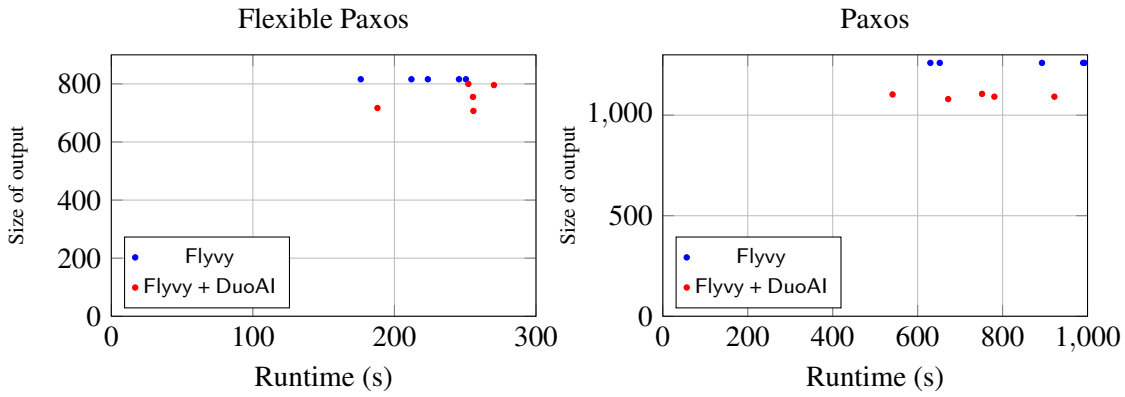


Fig. 5: Optimising Flyvy via DuoAI's output.

B Additional Statistics of Sec. 5.3

As said in Sec. 5.3, there are only two non-trivial protocols in Flyvy's benchmark that have quantifier alternation, for which we can produce comparable results. For both protocols, we execute the original Flyvy and the optimised Flyvy for 5 times with the sizes of the output (been discussed) and the runtime shown in Fig. 5. Performance-wise, we did not achieve notable improvement on the runtime of Flyvy, which is not unexpected, since the bottleneck of Flyvy is not the synthesis; instead, the SMT solving in Flyvy dominates the runtime and makes the runtime unstable.

Picat Through the Lens of Advent of Code

Neng-Fa Zhou
CUNY Brooklyn College and Graduate Center
nzhou@acm.org

Cristian Grozea*
Fraunhofer Institute FOKUS
Berlin, Germany
cristian.grozea@fokus.fraunhofer.de

Håkan Kjellerstrand
Independent Researcher, hakank.org
hkjellerstrand@acm.org

Oisín Mac Fhearaí
denpashogai@gmail.com

Picat is a logic-based, multi-paradigm programming language that integrates features from logic, functional, constraint, and imperative programming paradigms. This paper presents solutions to several problems from the 2024 Advent of Code (AoC). While AoC problems are not designed for any specific programming language, certain problem types, such as reverse engineering and path-finding, are particularly well-suited to Picat due to its built-in constraint solving, pattern matching, backtracking, and dynamic programming with tabling. This paper demonstrates that Picat's features, especially its SAT-based constraint solving and tabling, enable concise, declarative, and highly efficient implementations of problems that would require significantly more effort in imperative languages.

1 Introduction

The Advent of Code (AoC) is an annual programming challenge that presents a series of algorithmic problems[8], attracting programmers from diverse backgrounds worldwide. The problems range from simple puzzles to complex computational challenges, making them an excellent benchmark for evaluating programming languages and problem-solving paradigms. A defining feature of AoC is that each daily challenge has a first part, that is usually easier to solve, and a second part, that comes after a twist of the original problem proposed in the first part, making it generally much more difficult to solve and requiring flexibility such that the solution or at least the model defined for the first part can be reused.

Picat [17], a logic-based multi-paradigm programming language, offers a unique combination of logic, functional, constraint, and imperative programming styles. It is particularly well-suited for solving AoC problems due to its built-in constraint solving, pattern matching, backtracking, and dynamic programming with tabling features. These capabilities allow for concise, declarative, and highly efficient implementations of problems that would require significantly more effort in imperative languages.

In this paper, we explore Picat solutions to five selected problems from the 2024 Advent of Code, demonstrating how Picat's distinctive features provide elegant and efficient solutions. While there is work on the usage of Picat for various kinds of problems [3, 11], we demonstrate in this paper some of the new features of Picat added over the past decade. Specifically, we apply Picat's SAT-based constraint solver to solve three combinatorial problems, showcasing its ability to efficiently model and solve constraint satisfaction problems. Additionally, we leverage tabling to solve two problems that require dynamic programming, illustrating how Picat's built-in tabling mechanism simplifies recursive computations and optimizes performance.

*Cristian Grozea's work has been partially funded by the German Federal Ministry of Education and Research (BMBF, Germany) as part of the 6G Research and Innovation Cluster 6G-RIC under Grant 16KISK020K.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 528–541, doi:10.4204/EPTCS.439.36

© Zhou, Grozea, Kjellerstrand, Mac Fhearaí
This work is licensed under the
Creative Commons Attribution License.

This paper presents Picat solutions for the following problems: **Day 23: LAN party**, a clique-finding graph problem; **Day 17: Chronospatial Computer**, a reverse-engineering puzzle; **Day 24: Crossed Wires**, a circuit simulation and repair problem; **Day 16: Reindeer Maze**, a path-finding problem; **Day 21: Keypad Conundrum**, an optimal planning problem. Through these case studies, we aim to highlight the expressiveness and power of Picat's constraint solving and tabling features, demonstrating its advantages in tackling complex algorithmic challenges efficiently. For each problem, we also give a brief comparison with other solutions in terms of efficiency and needed efforts. All the CPU times reported in this paper were measured on an Ubuntu Linux machine with an Intel i9-7940X CPU @ 3.10GHz and 64Gb RAM. For SAT-based solutions, Kissat[1] is used.

2 The Picat Language

This section gives an overview of the features of the Picat language. Readers are referred to [13] for a detailed description of the language, and [18] for uses of Picat in modeling and solving combinatorial problems.

Picat is a logic-based multi-paradigm programming language designed for both general-purpose scripting and combinatorial problem modeling. It is grounded in foundational logic programming technologies, including the design of Prolog [2, 5], the Warren Abstract Machine (WAM) [6, 1], Constraint Logic Programming over Finite Domains (CLP(FD)) [4], and tabling [7].

Picat supports a rich set of data types, many of which go beyond those traditionally found in Prolog. While it includes standard types such as atoms, numbers, variables, and compound terms (shared with Prolog), Picat introduces several new data types that enhance usability and expressiveness, including mutable arrays, maps (aka dictionaries), and sets.

In Picat, pattern-matching rules are used to define predicates and functions. Picat has two types of pattern-matching rules: the *non-backtrackable* rule

$Head, Cond$ => $Body$.

and the *backtrackable* rule

$Head, Cond$?=> $Body$.

For a call, a rule is applicable if the pattern in $Head$ matches the call and the condition $Cond$ is true. Once an applicable rule is found, Picat executes $Body$. If the selected rule is non-backtrackable, Picat commits to this rule, meaning that no other rules will be considered, even if $Body$ fails. If the selected rule is backtrackable, Picat backtracks when $Body$ fails, and continues to search for the next applicable rule. Picat also supports Prolog-style Horn clauses.

Picat distinguishes between functions, which return values, and predicates, which define relations and may have multiple solutions. A predicate head has the form $p(T_1,\ldots,T_n)$, where p is a predicate symbol and each T_i is an argument pattern. A function head has the form $f(T_1,\ldots,T_n) = R$, where f is a function symbol, each T_i is an argument pattern, and R is the return value. If both the condition $Cond$ and the body $Body$ are true, the rule can be abbreviated as $f(T_1,\ldots,T_n) = R$. Unlike Prolog, where a structural term automatically constructs a compound term, Picat uses the `$` symbol before a structural term to explicitly denote structure construction. So while `max(X,Y)` is a function call, `$max(X,Y)` is a structure construction.

Picat supports list and array comprehensions, enabling concise construction of complex data structures through declarative expressions. For example, the following array comprehension reads a maze map as input and converts it to a two-dimensional array:

[1] `https://github.com/arminbiere/kissat`

```
M = {to_array(L) : L in read_file_lines(File)}
```

Picat provides loops to support imperative-style programming alongside its declarative features. A `foreach` loop has the following general form:

```
foreach (E1 in D1, Cond1, ..., En in Dn, Condn)
    Goal
end
```

The expression E_i `in` D_i is called an *iterator*, where E_i is an iterating pattern, and D_i is an expression that gives a compound value. Each $Cond_i$ is an optional condition on iterators E_1 through E_i. It is used for looping through collections like lists, arrays, and ranges. Picat compiles loops away by transforming them into new tail-recursive predicate definitions [14].

Picat supports the assignment operator, which is particularly useful in loops where a variable's value needs to be updated over time. An assignment in the form $LHS := Exp$ assigns the value of Exp to LHS. If LHS is an indexing operation of the form $X[I_1,\ldots,I_n]$, Picat updates the value stored at the specified location within the indexed structure. If LHS is a variable, Picat introduces a new variable to hold the value of Exp and replaces all subsequent occurrences of the original variable with this new one. In this latter case, where LHS is a variable, Picat compiles the assignment away by transforming it into new predicate definitions [14].

These new data types, combined with Picat's support for pattern matching, comprehensions, and imperative constructs, make the language more expressive and practical for a wide range of programming tasks, especially those involving structured data and numeric computation.

Picat provides a unified interface for CP, SAT, MIP, and SMT solvers, allowing seamless solver switching for constraint satisfaction problems. It includes an award-winning SAT-based CSP solver, PicatSAT[15, 16], with efficient encodings for high-level constraints.

Picat offers a tabling mechanism for dynamic programming solutions. It incorporates structure sharing for tabled data to efficiently manage memory and avoid state explosion [10]. The tabling-based planner of Picat provides a number of strategies for efficiently solving planning problems using state-space search [12].

These features make Picat a powerful tool for both general-purpose programming and combinatorial problem modeling.

3 Day 23: LAN Party

In Advent of Code 2024, Day 23, the problem revolves around analyzing a network of computers connected by bidirectional links. The input describes these connections, forming an undirected graph, and the goal is to extract meaningful structural properties from this network.

Part-1 is to identify all unique triads—sets of three computers that are all directly connected to one another. Additionally, the problem requires counting how many of these triads include at least one computer whose name starts with 't'. This part can be trivially solved in any programming language by counting triads involving each of the vertices.

Part-2 escalates the challenge by requiring the identification of a maximum clique in the network. A clique is a subset of nodes in which every pair of nodes is directly connected, and the maximum clique is the largest such subset. Finding a maximum clique is an NP-hard problem, making it a suitable challenge for constraint programming and graph algorithms.

Our Picat solution is given below.

```
% A Picat program for the maximum clique problem.

import sat.

main([File]) =>
    input(File,VSet,EdgeSet),
    maxClique(VSet,EdgeSet,Bs),
    output(VSet,EdgeSet,Bs).

maxClique(VSet,EdgeSet) =>
    Vs = to_array(keys(VSet)),
    N = len(Vs),
    Bs = new_array(N),
    Bs :: 0..1,
    foreach (I in 1..N-1, J in I+1..N)
        if not EdgeSet.has_key([Vs[I],Vs[J]]) then
            #~Bs[I] #\/ #~Bs[J]
        end
    end,
    Count #= sum([B : B in Bs]),
    solve([$max(Count)],Bs).
```

The first line imports the `sat` module. Picat supports four solver modules: `sat`, `cp`, `mit`, and `smt`. The import statement specifies that the SAT-based solver should be used for this program.

The `input(File,VSet,EdgeSet)` predicate reads a graph specification from `File` into `VSet` and `EdgeSet`, where `VSet` is a hash set of vertices, and `EdgeSet` is a hash set of edges.

The `maxClique(VSet,EdgeSet)` predicate formulates the maximum clique problem as a constraint optimization problem. It assigns a binary variable to each vertex, where 1 indicates the vertex is included in the clique and 0 otherwise. For any two vertices `I` and `J` that are not connected by an edge, let `Bs[I]` and `Bs[J]` be their corresponding binary variables. The model enforces that these two unconnected vertices cannot both be included in the clique using the constraint: `#~Bs[I] #\/ #~Bs[J]`. The objective is to maximize the sum of the binary variables.

It took the Picat program 10.8 seconds to find a maximum clique in the input graph, which is made of 520 vertices and 3380 edges. When Kissat is switched to a max-sat using CGSS2[2], the program took 2.01 seconds. Some of the solutions in other languages on Reddit[3] are much faster than our SAT-based solution. For example, the solution based on the Bron-Kerbosch algorithm, a recursive backtracking algorithm, only took 0.055 seconds. Even the brute force solution[4], which recursively expands cliques, took the same amount of time. While our solution is not competitive on the input graph in terms of speed, it is a straightforward declarative solution that can be written without knowledge of specialized graph algorithms.

[2]To switch to maxsat, change `solve([$max(Count)],Bs)` to `solve([$max(Count),maxsat],Bs)`.

[3]`https://www.reddit.com/r/adventofcode/comments/1hkgj5b/2024_day_23_solutions/`

[4]`https://github.com/tmo1/adventofcode/blob/main/2024/23b.py`

4 Day 17: Chronospatial Computer

In Advent of Code 2024, Day 17, participants are tasked with analyzing a program designed for an imaginary machine called a Chronospatial Computer (CC). This CC computer operates with three registers A, B, and C, which can each hold any integer value. Its program consists of a sequence of 3-bit numbers (ranging from 0 to 7), each representing either an instruction opcode or an operand. For example, the following is a CC program, where the left column gives instructions and the right column gives the representing symbols.

2,4	`B = A mod 8`
1,5	`B = B^ 5`
7,5	`C = A/2**B`
0,3	`A = A/2**3`
4,0	`B = B ^ C`
1,6	`B = B ^ 6`
5,5	`out B mod 8`
3,0	`jnz 0`

The last instruction `"jnz 0"` sets the instruction pointer to the first instruction if register A contains a non-zero value.

The challenge is divided into two parts, focusing on different aspects of program execution and reverse engineering. Part-1 requires careful implementation of the instruction set to accurately trace the program's behavior and determine the resulting output sequence. Part-2 presents a reverse engineering challenge. Given a specific output sequence, participants must deduce the smallest possible initial value of register A that, when the program is executed, produces the program itself as the output. This involves understanding the relationship between the initial state and the program's output, effectively working backward from the output to infer the input. As the output is the same as the program itself, the sequence of instructions up to `"out B mod 8"` needs to be executed 16 times before register A holds the value 0 and the program terminates.

Our Picat solution for the above CC program is given below.

```
% A Picat program for Chronospatial Computer reverse engineering.

import sat.

main =>
    A = new_bv(56),
    gen(A,[2,4,1,5,7,5,0,3,4,0,1,6,5,5,3,0]),
    solve($[min(A)],A),
    println(bv_to_int(A)).

gen(A,[]) => bv_eq(A,0).          % halt when A = 0
gen(A,[Z|Zs]) =>
    B = bv_take(A,3),             % B = A mod 8
    bv_xor(B,5,B1),               % B1 = B^5
    bv_pow(2,B1,P),               % P = 2**B1
    bv_div(A,P,C),                % C = A div P
    A1 = bv_drop(A,3),            % A1 = A/2**3
    bv_xor(B1,C,B2),              % B2 = B1^C
```

```
bv_xor(B2,6,B3),              % B3 = B2^6
Z #= 4*B3[3]+2*B3[2]+B3[1],   % out B3 mod 8
gen(A1,Zs).
```

It encodes the CC program into a constraint network using bit vector and integer-domain constraints, and solves the constraints while minimizing the input. The `main` predicate creates a 56-bit bit-vector `A`, invokes `gen` to encode the sequence of instructions as constraints, calls `solve` to find a solution that minimizes `A`, and finally outputs the resulting value of `A`.

The `gen` predicate takes a bit vector representing the value in register `A` and a list of instructions from the given CC program. When the list is empty, the constraint `bv_eq(A,0)` ensures that `A` is 0. Since the program generates a copy of itself, the sequence of instructions up to instruction `"out B3 mod 8"` executes once per code in the CC program.

Our Picat program took about 1 second to find the minimum input `109019476330651`. In terms of speed, our solution is not as fast as the DFS-based solution[5], which only took 0.006 seconds, or the naive iterative solution [6], which took 0.07 seconds. However, our solution is arguably more elegant than those solutions. Moreover, it could be more scalable than those solutions for large inputs.

5 Day 24: Crossed Wires

In Advent of Code 2024, Day 24, titled "Crossed Wires", the participants are tasked in the part one with modeling a Boolean circuit built from three types of logic gates (AND, OR and XOR). One example of such gate instance is given in Equation 1. In this example, the wires named *abc* and *x*32 are the inputs to the gate and the wire def is the output of the gate.

$$\text{abc XOR x32} \longrightarrow \text{def} \tag{1}$$

Part-1 requires modeling one single evaluation of a given circuit on a given input. This task can be easily tackled with any programming languages.

Part-2 specifies that the circuit is meant to be a binary addition circuit. More precisely, given the problem input, it should compute $z = x + y$, where $x = x_{44}..x_0$ is a binary number of 45 bits, where x_0 is the least significant bit, $y = y_{44}..y_0$ is another 45-bit number, and the result is $z = z_{44}..z_0$. However, the circuit is faulty, with the outputs of four pairs of gates swapped. The task is to identify which gates are involved in those swaps, and swap the outputs back so that the circuit functions correctly for every inputs.

The complexity of the problem arises from the presence of $n > 300$ gates in the circuit. It is infeasible to solve the problem by brute force, as one would need to try the number of combinations given in Equation 2.

$$\frac{n!}{2^4 \cdot 4! \cdot (n-8)!} > 10^{17} \tag{2}$$

This problem can be modeled as a constraint satisfaction problem. A straightforward model uses a binary variable for each pair of distinct gates, which indicates if the pair is involved in a swap. However, this model is inefficient as it requires 44000 binary variables.

Our model treats the problem as an assignment problem. It uses 8 slots, numbered 1 through 8, assuming slots i and $i+1$ are involved in a swap, for $i \in \{1,3,5,7\}$, and maps the slots to gates. To figure

[5] https://www.reddit.com/r/adventofcode/comments/1hg38ah/comment/mb3vqjy/
[6] https://www.reddit.com/r/adventofcode/comments/1hg38ah/comment/m406q9f/

out the mapping, one can use one binary variable for each slot and each gate. In our Picat solution,[7] presented below, an integer-domain variable is used for each slot, which maps the slot to a distinct gate.

```
% The core of the Picat program for Crossed Wires.

main([File])=>
    read_input(File,Gates),
    N = len(Gates),
    Slots = new_array(8),
    Slots :: 1..N,                          % each slot is mapped to a distinct gate
    all_different(Slots),
    NInputs = 40,                           % train the circuit using 40 random inputs
    foreach(_ in 1..NInputs)
        Vx := {random(0,1) : _ in 1..45},   % random input for X
        Vy := {random(0,1) : _ in 1..45},   % random input for Y
        Vz := new_array(45),
        Vz :: 0..1,                         % output Z
        add_gate_constrs(Gates,Slots,Vx,Vy,Vz),
    end,
    solve(Slots),
    output(Slots, Gates).
```

The program takes the input file name `File` as a command line argument. It reads `Gates` from the file. It uses an array of 8 integer-domain variables, each of which has the domain `1..N`, where `N` is the number of gates. The `all_different` constraint ensures that each slot is mapped to a distinct gate.

The program generates a number of training inputs and calls `add_gate_constrs` to ensure that the circuit produces the expected output with the swaps. It is generally difficult to estimate in advance the number of training inputs needed. We need to run the program with a number of inputs that is large enough to produce a unique answer.

For each training input, a binary variable is created for each wire. The `add_gate_constrs` predicate converts the gates to constraints. For example, for the gate in 1, it generates the constraint

```
Babc #^ Bx32 #= Bdef
```

where the binary variable's name of each wire is the wire's name preceded with the letter `B`. For the very first layer of the circuit, the input wires are not involved in any swaps, so the values are taken from `Vx` and `Vy`. For any other wire of a gate, we need to determine its binary variable based on whether or not the gate is involved in a swap. If the gate is not involved in any swaps, we use the variable of the wire itself. Otherwise, we need to use its partner wire's binary variable. The `add_gate_constrs` predicate also ensures that adding `Vx` and `Vy` produces `Vz`, which can be forced by using the bit-vector addition constraint `bv_add(Vx,Vy,Vz)`.

The statistics shows that the Day 24 problem is the toughest AoC problem in 2024. Our model is encoded into SAT with 996000 clauses and 361000 variables. Despite the large encoding size, it is solved in 3.6 seconds. This is a testimony to the competitiveness of the Picat's SAT encoder and the Kissat solver. The straightforward model that uses a binary variable for each pair of gates fails to solve the problem in 2 hours.

[7] `https://github.com/cgrozea/AdventOfCode2024/blob/main/24/cleanup/part2article.pi`

Many of the approaches available on Reddit for Day 24 [8] required manual inspection of the problem instance, and can't solve all instances that follow the problem description (see our instances generator[9]). The program by `ayoubzulfiqar`[10] for the problem is much faster than our SAT-based solution (it solves the problem in 0.06 seconds). It performs the ripple-carry algorithm on the inputs and identify the potential wires that could be swapped. While the solution is very intelligent and efficient, it requires genius insights and is limited by the assumption that the circuit follows a standard pattern repeated for every bit position.

6 Day 16: Reindeer Maze

In Advent of Code 2024, Day 16, participants are presented with a challenge titled "Reindeer Maze," which involves navigating a complex maze to guide a reindeer from a starting point to a destination. The problem is divided into two parts, each introducing unique objectives and constraints.

Part-1 requires participants to determine the minimum score needed for the reindeer to reach the destination. The maze is represented as a grid, which the reindeer can traverse in any of the four cardinal directions – north, east, south and west – as well as rotate left or right by 90 degrees. The objective is to find the path from the starting point to the destination that yields the lowest cumulative score. One slight complication is that orientation must be tracked to produce the correct result – that is, if the reindeer is facing east and wants to move south, it must first turn clockwise, then move forwards. Turning has a significantly greater cost than moving forwards, so an optimal path minimizes the number of rotations. The following path-finding program using tabling can be easily adapted for this problem:

```
table(+,+,-,min)
path(State,State,Path,Cost) =>
    Path = [],
    Cost = 0.
path(State,Goal,Path,Cost) =>
    Path = [(State,NextState)|Path1],
    edge(State,NextState,ThisCost),
    path(NextState,Goal,Path1,Cost1),
    Cost = ThisCost+Cost1.
```

The predicate `edge(State,NextState,Cost)` specifies a weighted directed graph, where `Cost` is the weight of the edge from `State` to `NextState`. The predicate `path(State,Goal,Path,Cost)` finds a path `Path` from `State` to `Goal` with the minimum cost. The table mode denotes that the first two arguments are inputs, the third one is an output, and the last argument is to be minimized, meaning that for each input pair, the system stores only one path with the minimum cost.

Part-2 extends the challenge by asking participants to identify all nodes that lie on *any* of the shortest paths from the start to the destination. This requires not only finding the minimum score but also determining which nodes are part of paths that achieve this score.

An easy approach is to modify the above example, letting a state carry a via point. A node N is on a shortest path if the shortest path cost from the start node to the destination with N as a via point is globally minimum. However, this approach is inefficient as it significantly increases the number of states to be explored.

[8] `https://www.reddit.com/r/adventofcode/comments/1hl698z/2024_day_24_solutions/`

[9] `https://github.com/cgrozea/AdventOfCode2024/blob/main/24/gen.pi`

[10] `https://github.com/ayoubzulfiqar/advent-of-code/tree/main/2024/Python/Day24`

A strategic approach involves a combination of reverse traversals from the destination back to all intermediate nodes together with forward traversals from the start node as follows:

1. **Forward traversals:** Determine the move costs C_{SN} from the start node S to each reachable node N in the grid.
2. **Backward traversals:** Determine the reverse move costs C_{ND} from the destination node D to each reachable node N in the grid, in each of the four possible orientations (since it may be possible to arrive at the destination from multiple directions).
3. **Count nodes on optimal paths:** Find all unique nodes that belong to an optimal path by searching the cost table for each node N satisfying the equation $C_{SN} + C_{ND} = MinCost$, where $MinCost$ is the minimum path cost found in Part-1.

The following shows an implementation of the approach using tabling:[11]

```
% A Picat program using tabling for Reindeer Maze.

main([File]) =>
    read_input(File,M,NRows,NCols,RStart,CStart,REnd,CEnd),
    Count = 2,
    foreach (R in 2..NRows-1, C in 2..NCols-1,
             (R,C) != (RStart,CStart), (R,C) != (REnd,CEnd), M[R,C] !== '#')
        if minof(best_path_via(R,C,M,RStart,CStart,REnd,CEnd,Cost),Cost), Cost == 94444 then
            Count := Count+1,
        end
    end,
    println(Count).

% the best path from (RStart,CStart) to (REnd,CEnd) via (R,C)
best_path_via(R,C,M,RStart,CStart,REnd,CEnd,Cost) =>
    member([Dr,Dc], [[-1,0],[1,0],[0,-1],[0,1]]),
    best_path_to_start([Dr,Dc,R,C,M,RStart,CStart],Cost1),
    best_path_to_end([-Dr,-Dc,R,C,M,REnd,CEnd],Cost2),  % change direction by 180 degree
    Cost = Cost1+Cost2.

table (+,min)
best_path_to_start([Dr,Dc,R,C,M,R,C],Cost), Dr == 0, Dc == -1 =>  % direction must be west (0,-1)
    Cost = 0.
best_path_to_start([Dr,Dc|T],Cost) ?=>                          % rotate
    turn(Dr,Dc,Dr1,Dc1),
    best_path_to_start([Dr1,Dc1|T],Cost1),
    Cost = Cost1+1000.
best_path_to_start([Dr,Dc,R,C,M|T],Cost) =>                     % move
    R1 = R+Dr,
    C1 = C+Dc,
    M[R1,C1] != '#',
    best_path_to_start([Dr,Dc,R1,C1,M|T],Cost1),
    Cost = Cost1+1.
```

[11] The full source is available at: https://github.com/nfzhou/aoc/blob/main/aoc_24_16_part2.pi

```
table (+,min)
best_path_to_end([Dr,Dc,R,C,M,R,C],Cost) => Cost = 0.
best_path_to_end([Dr,Dc|T],Cost) ?=>                          % rotate
    turn(Dr,Dc,Dr1,Dc1),
    best_path_to_end([Dr1,Dc1|T],Cost1),
    Cost = Cost1+1000.
best_path_to_end([Dr,Dc,R,C,M|T],Cost) =>                     % move
    R1 = R+Dr,
    C1 = C+Dc,
    M[R1,C1] != '#',
    best_path_to_end([Dr,Dc,R1,C1,M|T],Cost1),
    Cost = Cost1+1.
```

The `read_input(F,M,NRows,NCols,RStart,CStart,REnd,CEnd)` predicate reads the maze from `File` into matrix `M`, where `NRows` is the number of rows of `M`, `NCols` is the number of columns of `M`, `(RStart,CStart)` is the starting node, and `(REnd,CEnd)` is the destination.

The `main` predicate uses a loop to count the number of nodes on the shortest paths. It initializes `Count` to 2, counting in the starting and destination nodes. For each node `(R,C)` that is not the starting node, the destination, or an obstacle, if the cost of the best path from the starting node to the destination via node `(R,C)` equals the shortest-path's cost (94444, the result of Part-1), it increments `Count`.

The `best_path_via(R,C,M,RStart,CStart,REnd,CEnd,Cost)` predicate finds the best-path cost from `(RStart,CStart)` to `(REnd,CEnd)` via `(R,C)`. In order to facilitate sharing for tabled calls, it finds the best path cost from `(R,C)` to `(RStart,CStart)`, and the best path cost from `(R,C)` to `(REnd,CEnd)`. It uses deltas to represent directions. For example, `(Dr,Dc) = (0,-1)` represents the westward direction. In order to ensure that the forward and backward paths can be connected to form a viable path, it starts the two searches with opposite initial directions. The `best_path_to_start` and `best_path_to_end` are defined in the same way as the `path` predicate given above. The goal condition for `best_path_to_start` also ensures that the direction is west, as the Reindeer faces east in the beginning.

As the `best_path_via` may return multiple shortest path costs due the nondeterministic choice for the starting direction for the backward search[12], the program uses `minof` to retrieve the minimum of such path costs.

The implementation using tabling took 16 seconds to find the count 502 for the input maze. We also implemented the approach in Picat using Dijkstra's algorithm[13]. This implementation only took 1 second to find the answer.

7 Day 21: Keypad Conundrum

In Advent of Code 2024, Day 21, titled "Keypad Conundrum," participants are tasked with navigating a series of interconnected keypads to input specific codes, with each part of the challenge introducing additional layers of complexity.

In Part-1, the initial scenario involves a directional keypad that the human controller uses (Figure 1). The objective is to determine the shortest sequence of button presses required to instruct a robot to type a given code, such as '029A', on a numeric keypad (Figure 1). The robot starts with its arm positioned over the 'A' button on the numeric pad. Each directional button moves the robot's arm in a specific direction

[12]The call `member([Dr,Dc], [[-1,0],[1,0],[0,-1],[0,1]])` is nondeterministic.

[13]`https://github.com/DestyNova/advent_of_code_2024/blob/main/16/part2_dijkstra.pi`

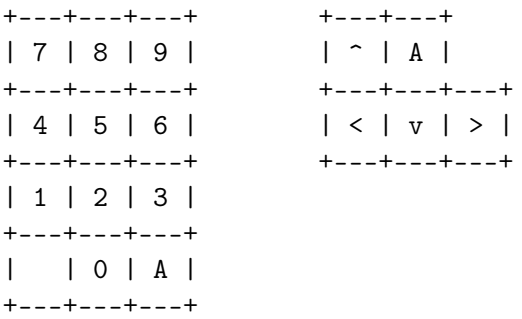


Figure 1: The numeric (left) and directional (right) keypads.

and the activate button (A) instructs the robot to press the current button. The challenge is to compute the minimal sequence that results in the robot correctly inputting the desired code on the numeric keypad. Part-1 can be solved with brute-force search.

In Part-2, the complexity increases as additional layers are introduced. Now, a second robot is required to control the first robot, a third robot is required to control the second robot, so on up to the 26th robot, and human controller instructs the 26th robot. The task is to determine the shortest sequence of button presses for the human controller that will propagate through the chain of robots, ultimately resulting in the correct code being entered on the numeric keypad.

The output is the sum of the complexities of the codes, where each code's complexity is equal to the result of multiplying the length of the shortest sequence for the human controller and the numeric part of the code.

The following shows the main part of a Picat program for this problem.[14]

```
% A Picat program using tabling for Keypad Conundrum.

main([File]) =>
    Codes = read_file_lines(File),
    Comp = 0,
    foreach (Code in Codes)
        num_pad_plan(t,'A',Code,Plan,_),
        trans_plan_rec(25,Plan,Len),
        Comp := Comp + to_int([D : D in Code, digit(D)]) * Len
    end,
    println(Comp).

table (+,+,+,-,min)
num_pad_plan(PreDir,_,[],Plan,Obj) =>
    Plan = [],
    Obj = (0, 0).
num_pad_plan(PreDir,C,[C|Code],Plan,Obj) =>
    Plan = ['A'|Plan1],
    num_pad_plan(t,C,Code,Plan1,Obj1),
    Obj1 = (Len1, Turn1),
    Obj = (Len1+1,Turn1).
```

[14]The full source is available at: `https://github.com/nfzhou/aoc/blob/main/aoc_24_21_part2.pi`

```
num_pad_plan(PreDir,C,Code,Plan,Obj) =>
    member(Dir,[l,u,d,r]),   % best ordering
    move_num_pad(Dir,C,NextC),
    Plan = [Dir|Plan1],
    num_pad_plan(Dir,NextC,Code,Plan1,Obj1),
    Obj1 = (Len1, Turn1),
    Obj = (Len1+1,cond(Dir==PreDir,Turn1,Turn1+1)).

table
trans_plan_rec(0,Plan,Len) =>
    Len = len(Plan).
trans_plan_rec(N,[],Len) => Len = 0.
trans_plan_rec(N,Plan,Len) =>
    extract_chunk(Plan,PlanR,Chunk),
    dir_pad_plan(t,'A',Chunk,ChunkPlan,_),
    trans_plan_rec(N-1,ChunkPlan,Len1),
    trans_plan_rec(N,PlanR,Len2),
    Len is Len1+Len2.
```

The `main` predicate reads codes from the input file and iterates over them using a `foreach` loop. For each code, the predicate `num_pad_plan` computes an optimal plan to instruct the numeric pad robot to generate the code. This plan is then recursively translated through 25 control layers by the predicate `trans_plan_rec`, producing a final plan for the human controller. An assignment statement computes the code's complexity and adds it to the variable `comp`, which accumulates the total complexity.

The predicate `num_pad_plan(PreDir,C,Code,Plan,Obj)` takes three inputs: `PreDir`, the previous direction of movement of the numeric robot; `C`, the robot's current key position; and `Code`, the sequence to be typed. It produces a plan `Plan` that minimizes the objective value `Obj`, which is a pair `(Length, Turns)` representing the plan's length and number of turns. The table mode declaration `(+,+,+,-,min)` specifies that the first three arguments are inputs, the last two are outputs, and `Obj` is to be minimized. Initially, the robot starts at key 'A' with a dummy direction `t` passed to `PreDir`. The definition of `num_pad_plan` is straightforward:

The first rule handles the base case where the code is empty. It returns an empty plan and the objective value `(0, 0)`.

The second rule applies when the robot is already at the correct key `C`. It adds 'A' to the plan to simulate pressing the key, and recursively calls itself to find an optimal plan `Plan1` and objective `Obj1` for the remaining code. The resulting `Obj` is derived by incrementing the plan length while preserving the number of turns.

The third rule handles the case where the robot is not on the required key. It nondeterministically chooses a direction in the order left (l), up (u), down (d), and right (r), then recursively computes a subplan `Plan1` and objective `Obj1` after the robot moves in that direction. The final objective `Obj` is calculated by incrementing both the length and the number of turns.

The predicate `trans_plan_rec(N, Plan, Len)` translates a plan from the numeric robot up the control chain to the `N`th controller:

The base case, when `N = 0`, terminates the recursion.

The general case processes the plan in chunks, where a chunk is the longest subplan that begins with a non-'A' symbol, terminates with 'A' symbols, and contains no intermediate 'A' symbols. The predicate `extract_chunk(Plan, PlanR, Chunk)` splits `Plan` into `Chunk`, the first chunk, and `PlanR`,

the remaining sequence. The predicate `dir_pad_plan(t, 'A', Chunk, ChunkPlan, _)`—defined similarly to `num_pad_plan`—finds an optimal plan `ChunkPlan` for the next controller up the chain. The rule then recursively calls `trans_plan_rec` on `ChunkPlan` and `PlanR`, decrementing the level since `ChunkPlan` belongs to the upper controller.

The program computes the result 226179529377982 in just 0.037 seconds for the input list `["671A", "826A", "670A", "085A", "283A"]`. The solution leverages two key ideas. First, it limits direction choices to a fixed order [l, u, d, r] instead of trying all permutations, significantly reducing the search space. Second, it performs chunk-wise translation along the control chain, with optimal subplan contributing to an optimal overall solution. Additionally, chunking facilitates sharing of tabled calls, improving efficiency.

The Reddit's AoC 24 Day 21 Megathread[15] gives several solutions in other languages. Many approaches use dynamic programming to find a minimal path. The Python program by `YOM2_UB`[16] uses depth-first search together with memoization. It only took 0.031 seconds to solve the problem. The Rust program by `michelkraemer`,[17], which also uses depth-first search with memoization, only took 0.013 seconds. While our Picat solution is not competitive in terms of speed, it is certainly more readable than those solutions, thanks to the built-in dynamic programming feature.

8 Conclusion

This paper presents Picat solutions for five problems selected from the 2024 Advent of Code collection. The solution to the Maximum Clique problem is primarily pedagogical, illustrating how to model and solve a simple combinatorial problem using SAT. The solution to the Chronospatial Computer reverse engineering problem translates a program into a constraint network using bit-vector constraints, allowing the inference of inputs from known outputs. The solution to the Crossed Wires problem showcases not only the power of SAT encoding and solving but also the critical role of effective modeling. The idea of using eight integer-domain variables to identify the swapped wires proves to be particularly effective.

The solution to the Reindeer Maze problem demonstrates the strength of a strategic reformulation: converting a one-to-many shortest paths problem into a many-to-one problem. By reversing the graph, the approach greatly enhances sharing among tabled calls. The Keypad Conundrum solution illustrates tabled search involving the simultaneous optimization of two objectives.

While the Picat solutions may not always be the shortest or fastest compared to those found online, they are declarative, readable, and do not rely on deep insights or clever tricks.

Many of the selected problems have real-world relevance. For instance, the reverse engineering problem relates to program analysis and verification, while the Crossed Wires problem mirrors fault diagnosis and troubleshooting tasks. These examples show that Picat can be a viable tool for such applications, much like other formal methods tools. Path-finding and planning problems are ubiquitous in practice. The solutions to the Reindeer Maze and Keypad Conundrum problems demonstrate how declarative dynamic programming can be expressed naturally in Picat, without requiring the implementation of complex algorithms or heuristics.

[15] `https://www.reddit.com/r/adventofcode/comments/1hj2odw/2024_day_21_solutions/`
[16] `https://pastebin.com/rExTBYr6`
[17] `https://github.com/michel-kraemer/adventofcode-rust/blob/main/2024/day21/src/main.rs`

References

[1] Hassan Aït-Kaci (1991): *Warren's Abstract Machine: A Tutorial Reconstruction*. MIT Press.

[2] Alain Colmerauer & Philippe Roussel (1993): *The Birth of Prolog*. In John A. N. Lee & Jean E. Sammet, editors: *History of Programming Languages Conference (HOPL-II), Preprints, Cambridge, Massachusetts, USA, April 20-23, 1993*, ACM, pp. 37–52, doi:10.1145/154766.155362.

[3] Sergii Dymchenko & Mariia Mykhailova (2015): *Declaratively Solving Google Code Jam Problems with Picat*. In: *PADL*, pp. 50–57, doi:10.1007/978-3-319-19686-2_4.

[4] Pascal van Hentenryck (1989): *Constraint Satisfaction in Logic Programming*. MIT Press.

[5] Robert Kowalski (1979): *Logic for Problem Solving*. North Holland, Elsevier.

[6] David H. D. Warren (1983): *An Abstract Prolog Instruction Set*. Technical Note 309, SRI International.

[7] David S. Warren (1992): *Memoing for Logic Programs*. *Comm. of the ACM, Special Section on Logic Programming* 35, pp. 93–111, doi:10.1145/131295.131299.

[8] Eric Wastl (2015-now): *Advent of Code*. `https://adventofcode.com`. [Online; accessed January 7, 2026].

[9] Eric Wastl (2024): *Day 24: Crossed Wires*. `https://adventofcode.com/2024/day/24`. [Online; accessed January 7, 2026].

[10] Neng-Fa Zhou (2012): *The Language Features and Architecture of B-Prolog*. *Theory and Practice of Logic Programming* 12(1-2), pp. 189–218, doi:10.1017/S1471068411000445.

[11] Neng-Fa Zhou (2016): *Programming in Picat*. In: *Rule Technologies. Research, Tools, and Applications - 10th International Symposium, RuleML 2016, Stony Brook, NY, USA, July 6-9, 2016. Proceedings*, *Lecture Notes in Computer Science* 9718, Springer, pp. 3–18, doi:10.1007/978-3-319-42019-6_1.

[12] Neng-Fa Zhou, Roman Barták & Agostino Dovier (2015): *Planning as Tabled Logic Programming*. *Theory and Practice of Logic Programming* 15(4-5), pp. 543–558, doi:10.1017/S1471068415000216.

[13] Neng-Fa Zhou & Jonathan Fruhman (2013–2026): *A User's Guide to Picat*. `https://picat-lang.org`. [Online; accessed January 7, 2026].

[14] Neng-Fa Zhou & Jonathan Fruhman (2017): *Canonicalizing High-Level Constructs in Picat*. In: *PADL*, pp. 19–33, doi:10.1007/978-3-319-51676-9_2.

[15] Neng-Fa Zhou & Håkan Kjellerstrand (2016): *The Picat-SAT compiler*. In: *Practical Aspects of Declarative Languages: 18th International Symposium, PADL 2016, St. Petersburg, FL, USA, January 18-19, 2016. Proceedings 18*, Springer, pp. 48–62, doi:10.1007/978-3-319-28228-2_4.

[16] Neng-Fa Zhou & Håkan Kjellerstrand (2017): *Optimizing SAT Encodings for Arithmetic Constraints*. In: *CP*, pp. 671–686, doi:10.1007/978-3-319-66158-2_43.

[17] Neng-Fa Zhou, Håkan Kjellerstrand & Jonathan Fruhman (2013–2026): *Picat Programming Language*. `https://picat-lang.org`. [Online; accessed January 7, 2026].

[18] Neng-Fa Zhou, Håkan Kjellerstrand & Jonathan Fruhman (2015): *Constraint solving and planning with Picat*. *Springer Briefs in Intelligent Systems* 11, Springer, doi:10.1007/978-3-319-25883-6.

Most Probable Explanation in Probabilistic Answer Set Programming

Damiano Azzolini
University of Ferrara

Giuseppe Mazzotta
University of Calabria

Francesco Ricca
University of Calabria

Fabrizio Riguzzi
University of Ferrara

Title. Most Probable Explanation in Probabilistic Answer Set Programming

Authors. Damiano Azzolini, Giuseppe Mazzotta, Francesco Ricca, and Fabrizio Riguzzi

Reference. Damiano Azzolini, Giuseppe Mazzotta, Francesco Ricca, and Fabrizio Riguzzi. Most Probable Explanation in Probabilistic Answer Set Programming. Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025.

URL: `https://doi.org/10.24963/ijcai.2025/1006`

Keywords. Statistical Relational Artificial Intelligence, Answer Set Programming, Probabilistic Answer Set Programming, Quantified Answer Set Programming, Algebraic Model Counting.

Statement. The work the submission is based on has not been already presented at a major logic programming forum. The paper is included in IJCAI 2025 proceedings.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 542–545, doi:10.4204/EPTCS.439.37

© D. Azzolini, G. Mazzotta, F. Ricca & F. Riguzzi
This work is licensed under the
Creative Commons Attribution License.

Contribution

The paper considers the Most Probable Explanation task in Probabilistic Answer Set Programming. A probabilistic answer set program (PASP) [4] is an answer set program [3] extended with ground probabilistic facts [5] of the form $p_i :: a_i$. Each selection σ of probabilistic facts (call Σ the set of all selections, $|\Sigma| = 2^n$ where n is the number of ground probabilistic facts) identifies a *world* w_σ, i.e., an answer set program, composed by the deterministic rules of the PASP and the considered probabilistic facts. Each world may have 0 or more stable models but the credal semantics (that we consider in the paper) requires at *least one*. We denote with $AS(w)$ the set of answer sets of a world w. A selection σ and so a world w_σ is associated with a probability computed as $P(w_\sigma) = \prod_{a_i \in w_\sigma} p_i \prod_{a_i \notin w_\sigma} (1 - p_i)$ since probabilistic facts are considered independent. In this paper, we focus on the cautious and brave MPE tasks: given a PASP and a conjunction of ground literals e called *evidence*, the *cautious MPE* problem consists in finding the selection σ with the highest associated probability where e is entailed in every answer set of the corresponding world w_σ, i.e.:

$$\underline{\text{MPE}}(e) = \underset{\sigma \in \Sigma | \forall M \in AS(w_\sigma), M \models e}{\arg\max} P(\sigma),$$

while the *brave MPE* problem consists in finding the selection σ with the highest associated probability where the e is entailed in at least one answer set of the corresponding world w_σ, i.e.:

$$\overline{\text{MPE}}(e) = \underset{\sigma \in \Sigma | \exists M \in AS(w_\sigma), M \models e}{\arg\max} P(\sigma).$$

When the PASP is stratified (i.e., each world has a unique answer set) the cautious and brave MPE states coincide.

In this paper, we first provide a Two-level Algebraic Model Counting [8, 9] (2AMC) representation of the task, which is a generic framework to encode many tasks in Statistical Relational Artificial Intelligence [11]. Then, we consider the upper MPE problem and propose an ASP encoding to solve such a task. The idea is that we can encode a probabilistic fact $p :: a$ as:

$$\{a\}. \quad prob(w, id_a) :- a. \quad prob(\overline{w}, id_a) :- not\ a.$$

We converted probabilities into log-probabilities, so the probability of a world can be computed with summations instead of multiplications. Then, we add a constraint $:- not\ e$ that imposes that the evidence should be true. Lastly, we resort to weak constraints to find the selection with the highest associated probability, by adding $:\sim prob(W, Id)\ [-W, Id]$.

The lower MPE state requires more attention since the evidence must be entailed in every answer set of a candidate world. Thus, we resort to Answer Set Programming with Quantifiers, ASP(Q) [10], which extends ASP by introducing quantifiers over answer sets, making the modeling of problems across the entire Polynomial Hierarchy [13] as natural as modeling NP-hard problems in ASP. We proposed an ASP(Q) such that quantified answer sets are in one-to-one correspondence with selections that cautiously entail the evidence e and the optimal quantified answer sets correspond to lower MPE states.

In the experimental evaluation, we considered stratified and non stratified PASP. For stratified PASP (where each world has exactly one model), we considered the following existing solvers: ProbLog [12], cplint [2], aspmc [6], and plingo [7]. The first three adopt knowledge compilation, while the latter uses an ASP approach similar to the one we proposed in this paper. Then, for non stratified PASP, only PASTA [1], to the best of our knowledge, is able to solve such a task. For stratified datasets, we consider

all solvers and obtain that our ASP solution is substantially faster than other existing tools. Then, we considered non-stratified datasets and compare ASP (for brave MPE) and ASP(Q) (for cautious MPE) with the proposed 2AMC representation (implemented on top of aspmc) and with PASTA. We obtain that, also in this case, ASP + ASP(Q) outperform other approaches.

Relevance of the paper to logic programming. The paper is relevant to Logic Programming since it deals with Probabilistic Answer Set Programming and proposes a novel approach to compute the MPE states (brave and cautious) with 2ACM and with ASP and ASP(Q).

Significance of the results. Our results show that, surprisingly, general tools such as ASP and ASP(Q), outperform specific tools based on knowledge compilation specifically tailored for probabilistic logic programs. Furthermore, it shows yet another application area for ASP and ASP(Q).

References

[1] Damiano Azzolini, Elena Bellodi & Fabrizio Riguzzi (2023): *MAP Inference in Probabilistic Answer Set Programs*. In Agostino Dovier, Angelo Montanari & Andrea Orlandini, editors: *AIxIA 2022 – Advances in Artificial Intelligence*, Springer International Publishing, Cham, pp. 413–426, doi:10.1007/978-3-031-27181-6_29.

[2] Elena Bellodi, Marco Alberti, Fabrizio Riguzzi & Riccardo Zese (2020): *MAP Inference for Probabilistic Logic Programming*. *Theory and Practice of Logic Programming* 20(5), pp. 641–655, doi:10.1017/S1471068420000174.

[3] Gerhard Brewka, Thomas Eiter & Miroslaw Truszczynski (2011): *Answer set programming at a glance*. *Communications of the ACM* 54(12), pp. 92–103, doi:10.1145/2043174.2043195.

[4] Fábio Gagliardi Cozman & Denis Deratani Mauá (2020): *The joy of Probabilistic Answer Set Programming: Semantics, complexity, expressivity, inference*. *International Journal of Approximate Reasoning* 125, pp. 218–239, doi:10.1016/j.ijar.2020.07.004.

[5] Luc De Raedt, Angelika Kimmig & Hannu Toivonen (2007): *ProbLog: A Probabilistic Prolog and Its Application in Link Discovery*. In Manuela M. Veloso, editor: *20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, 7, AAAI Press, pp. 2462–2467.

[6] Thomas Eiter, Markus Hecher & Rafael Kiesel (2024): *aspmc: New frontiers of algebraic answer set counting*. *Artificial Intelligence* 330, p. 104109, doi:10.1016/j.artint.2024.104109.

[7] Susana Hahn, Tomi Janhunen, Roland Kaminski, Javier Romero, Nicolas Ruhling & Torsten Schaub (2024): *Plingo: A System for Probabilistic Reasoning in Answer Set Programming*. *Theory and Practice of Logic Programming*, pp. 1–34.

[8] Rafael Kiesel, Pietro Totis & Angelika Kimmig (2022): *Efficient Knowledge Compilation Beyond Weighted Model Counting*. *Theory and Practice of Logic Programming* 22(4), pp. 505–522, doi:10.1017/S147106842200014X.

[9] Angelika Kimmig, Guy Van den Broeck & Luc De Raedt (2017): *Algebraic Model Counting*. *Journal of Applied Logic* 22(C), pp. 46–62, doi:10.1016/j.jal.2016.11.031.

[10] Giuseppe Mazzotta, Francesco Ricca & Mirek Truszczynski (2024): *Quantifying over Optimum Answer Sets*. *Theory and Practice of Logic Programming* 23(4), pp. 948–964, doi:10.1017/S1471068424000395.

[11] Luc De Raedt, Kristian Kersting, Sriraam Natarajan & David Poole (2016): *Statistical relational artificial intelligence: Logic, probability, and computation*. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 10(2), pp. 1–189, doi:10.1007/978-3-031-01574-8.

[12] Dimitar Sht. Shterionov, Joris Renkens, Jonas Vlasselaer, Angelika Kimmig, Wannes Meert & Gerda Janssens (2015): *The Most Probable Explanation for Probabilistic Logic Programs with Annotated Disjunctions*. In Jesse Davis & Jan Ramon, editors: *24th International Conference on Inductive Logic Programming (ILP 2014)*, *Lecture Notes in Computer Science* 9046, Springer, Berlin, Heidelberg, pp. 139–153, doi:10.1007/978-3-319-23708-4_10.

[13] Larry J. Stockmeyer (1976): *The Polynomial-Time Hierarchy*. *Theoretical Computer Science* 3(1), pp. 1–22, doi:10.1016/0304-3975(76)90061-X.

ASPECT: Answer Set rePresentation as vEctor graphiCs in laTex

Alessandro Bertagnon
Department of Environmental and Prevention Sciences, University of Ferrara, C.so Ercole I D'Este, 32, Ferrara, Italy.
alessandro.bertagnon@unife.it

Marco Gavanelli
Department of Engineering, University of Ferrara, Via Saragat 1, Ferrara, Italy.
marco.gavanelli@unife.it

Reference

Alessandro Bertagnon, Marco Gavanelli, ASPECT: Answer Set rePresentation as vEctor graphiCs in laTex, Journal of Logic and Computation, Volume 34, Issue 8, December 2024, Pages 1580–1607. https://doi.org/10.1093/logcom/exae042

Public URL:
https://doi.org/10.1093/logcom/exae042

Keywords

Answer Set Programming, Answer Set Visualization, Vector Graphics, LaTeX

This submission is based on a previous paper presented at CILC 2023 (38th Italian Conference on Computational Logic), which is not a major international conference. Moreover, the current manuscript represents a significantly extended and extensively revised version of that original work. This version has only been published in a peer-reviewed journal (Journal of Logic and Computation) and has never been presented at any major international forum in the field of logic programming, such as ICLP, LPNMR, LOPSTR, or LPAR.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 546–548, doi:10.4204/EPTCS.439.38

© A. Bertagnon & M. Gavanelli
This work is licensed under the
Creative Commons Attribution License.

Extended Abstract

Logic programming is a declarative paradigm widely used in the field of Artificial Intelligence (AI), where it provides a natural way to model complex reasoning tasks. A notable example is Answer Set Programming (ASP), which stands out for its expressive syntax and the efficiency of its solvers. These features make ASP particularly well suited for educational contexts, such as teaching logic-based problem solving, while also enabling the development of advanced applications. However, the output produced by ASP solvers, typically sets of ground atoms, is often long, verbose, and difficult to interpret. This limitation is especially evident for students who are new to the language, but also experienced users working on large-scale or real-world problems would benefit from an intuitive visualization of results.

In this paper, we introduce ASPECT (Answer Set rePresentation as vEctor graphiCs in LaTeX), a sub-language of ASP that allows users to create graphical representations of answer sets in a simple and declarative way. ASPECT extends ASP with a set of graphical primitives that are translated into TikZ code, a vector graphics language used in LaTeX. This integration lets users produce high-quality diagrams that can be directly included in scientific papers, presentations, or teaching materials without having to write a single line of imperative code.

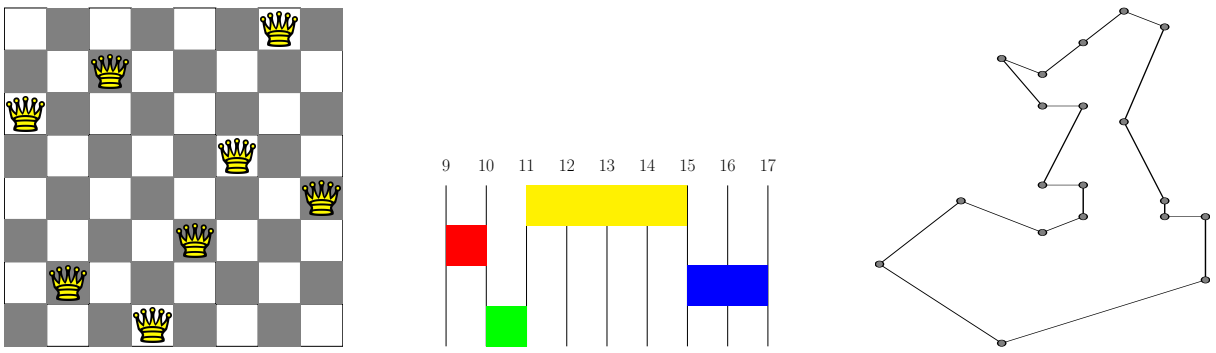
ASPECT provides a set of low-level drawing commands, such as rectangles, circles, ellipses, and lines, that resemble TikZ syntax and can be customized with colors and labels. For example, consider the well-known N-Queens problem, where the goal is to place N queens on an $N \times N$ chessboard so that no two queens threaten each other. Suppose the board is described in ASP with atoms `grid(I,J)`, and the queen positions are given by `queen(R,C)`. Using ASPECT, we can draw the board with a classic chequered pattern using the following rules:

```
aspect_fillrectangle(2*I-1,2*J-1,2*I+1,2*J+1,gray):- grid(I,J), I\2=J\2. %Dark squares
aspect_drawrectangle(2*I-1,2*J-1,2*I+1,2*J+1):- grid(I,J), I\2!=J\2. %Light squares
```

Each square is centered at `(2*I, 2*J)`, with alternating gray and white colors (the symbol `\` stands for modulus in ASP). To draw the queens, we use:

```
aspect_imagenode(2*R, 2*C, "queen.png", 50) :- queen(R,C).
```

Figure 1a shows the vector graphic produced by ASPECT with $N = 8$.



(a) (b) (c)

Figure 1: Three graphical representation drawn with ASPECT: (a) the N-Queens problem, (b) a scheduling or resource allocation task, (c) the Traveling Salesperson Problem (TSP).

ASPECT also supports animated graphics. Many ASP programs model dynamic processes, such as planning tasks, where a solution consists of a sequence of actions over time. ASPECT allows users to visualize these steps as individual frames. For example, in a robot motion domain:

```
aspect_drawcircle(X,Y,1,red,T):- robot_position(T,X,Y).
```

Here, each `aspect_drawcircle/5` atom draws a red circle at the robot's position for frame `T`, creating an animation of its movement. These frames can be compiled into Beamer presentations or JavaScript-driven PDFs containing vector graphics in motion.

ASPECT can also be used as a Domain-Specific Language (DSL) for graphics, independent of ASP's solving capabilities. In this context, logic rules are used not to compute solutions, but to describe figures and animations declaratively. For example, it is possible to define slides or algorithm visualizations entirely using ASP rules. This opens new opportunities in areas such as education, interactive explanations, and the creation of slides and diagrams.

Many tools have been developed in the past for visualizing answer sets, including *ASPViz* [2], *IDPDraw* [5], *Kara* [4], and *clingraph* [3]. These systems often require an additional mapping layer or external graphical interfaces. ASPECT, instead, uses a fully declarative approach where the logic and the visualization are described in the same ASP program. This makes it easier to learn and maintain.

Another strength of ASPECT is that it generates TikZ code directly. This means that the output can be used immediately in LaTeX documents, without any conversion or loss of quality. In addition, ASPECT supports animations, which most other systems do not, or only partially. It also runs entirely via command-line and requires no external graphical interface, making it portable and easy to integrate into automated workflows.

The contributions of this work are both practical and conceptual. Practically, ASPECT provides a simple and flexible language for building graphical representation and animations from ASP outputs. Conceptually, it shows how logic programming can be extended beyond problem solving to become a tool for visual communication. An open source implementation of the ASPECT interpreter is available (`https://github.com/abertagnon/aspect`), and it is also included in ASP Chef [1].

References

[1] Mario Alviano, Davide Cirimele & Luis Angel Rodriguez Reiners (2023): *Introducing ASP recipes and ASP Chef*. *CEUR Workshop Proceedings* 3437, CEUR-WS.org. Available at `https://ceur-ws.org/Vol-3437/paper4ASPOCP.pdf`.

[2] Owen Cliffe, Marina De Vos, Martin Brain & Julian A. Padget (2008): *ASPVIZ: Declarative Visualisation and Animation Using Answer Set Programming*. In: *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*, *Lecture Notes in Computer Science* 5366, Springer, pp. 724–728, doi:10.1007/978-3-540-89982-2_65.

[3] Susana Hahn, Orkunt Sabuncu, Torsten Schaub & Tobias Stolzmann (2022): *Clingraph: ASP-Based Visualization*. In: *Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings*, *Lecture Notes in Computer Science* 13416, Springer, pp. 401–414, doi:10.1007/978-3-031-15707-3_31.

[4] Christian Kloimüllner, Johannes Oetsch, Jörg Pührer & Hans Tompits (2011): *Kara: A System for Visualising and Visual Editing of Interpretations for Answer-Set Programs*. In: *Applications of Declarative Programming and Knowledge Management - 19th International Conference, INAP 2011, and 25th Workshop on Logic Programming, WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers*, *Lecture Notes in Computer Science* 7773, Springer, pp. 325–344, doi:10.1007/978-3-642-41524-1_20.

[5] Johan Wittocx (2009): *IDPDraw, a tool used for visualizing answer sets*. `https://wms.cs.kuleuven.be/dtai/pages/software/idpdraw`.

Probabilistic Traces in Declarative Process Mining

Michela Vespa
Dipartimento di Ingegneria
Università di Ferrara, Italy
michela.vespa@unife.it

Elena Bellodi
Dipartimento di Ingegneria
Università di Ferrara, Italy
elena.bellodi@unife.it

Federico Chesani
Dipartimento di Informatica - Scienza e Ingegneria,
Università di Bologna, Italy
federico.chesani@unibo.it

Daniela Loreti
Dipartimento di Informatica - Scienza e Ingegneria,
Università di Bologna, Italy
daniela.loreti@unibo.it

Paola Mello
Dipartimento di Informatica - Scienza e Ingegneria,
Università di Bologna, Italy
paola.mello@unibo.it

Evelina Lamma
Dipartimento di Ingegneria
Università di Ferrara, Italy
evelina.lamma@unife.it

Anna Ciampolini
Dipartimento di Informatica - Scienza e Ingegneria,
Università di Bologna, Italy
anna.ciampolini@unibo.it

Marco Gavanelli
Dipartimento di Ingegneria
Università di Ferrara, Italy
marco.gavanelli@unife.it

Riccardo Zese
Dipartimento di Scienze Chimiche, Farmaceutiche ed Agrarie
Università di Ferrara, Italy
riccardo.zese@unife.it

Vespa, M. et al. (2025). Probabilistic Traces in Declarative Process Mining. In: Artale, A., Cortellessa, G., Montali, M. (eds) AIxIA 2024 – Advances in Artificial Intelligence. AIxIA 2024. Lecture Notes in Computer Science 15450, Springer, Cham, pp. 330–345, doi:10.1007/978-3-031-80607-0_-25.

URL https://github.com/bovaristEliza/AIXIA2024

Keywords Process Mining, Probabilistic Compliance, Declarative Language, Distribution Semantics

Statement The work the submission is based on has not been already presented at a major logic programming forum.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 549–551, doi:10.4204/EPTCS.439.39

© M. Vespa et al.
This work is licensed under the
Creative Commons Attribution License.

1 Extended Abstract

In the Business Process Mining (BPM) community, the integration of probability into various aspects of process modeling is a growing research topic, reflecting an interest in enhancing process analysis under conditions of uncertainty in real-world domains. Uncertainty can be found at various levels, such as process constraints [1], events, event attributes, or traces.

The paper introduces a **novel formal semantics** for probabilistic process traces in declarative Process Mining (PM), inspired by the Distribution Semantics from Probabilistic Logic Programming (PLP). In fact, some information in a process trace could be unreliable and a domain expert might be allowed to indicate a degree of belief on that information. Such degree is expressed by an epistemic probability value which may be associated to an event execution.

We rely on PLP as, in a PLP program, logic formulas are annotated with probabilities, defining a probability distribution over normal logic programs called 'worlds'. Likewise, the probability of an event is treated as the probability that such event will appear (or not) in a process trace.

Example In a hospital department the log of the patients' events is usually updated by healthcare professionals. They might forget to log one or more events. A probabilistic trace would take in consideration the fact that, even if not logged, some events might have happened with a probability p. The trace:

$$t = \langle 0.7 : \mathit{check_fever}, \mathit{drug_prescription}, \mathit{check_fever}, 0.95 : \mathit{check_fever}, \mathit{dimission} \rangle$$

describes the situation where the healthcare professional logged only the second, the third and the fifth event. Two events of type "*check_fever*" were not logged, however it is probable that they happened anyway due to the patient's disease and treatment. In these cases we have a degree of belief 0.7 and 0.95 respectively in the occurrence of those events, i.e. we *believe* that those events happened with that probability. A probability value of 1 means the event happened, and we refer to it as "certain". A *probabilistic trace* is a trace where at least one event is probabilistic.

Secondly, the paper introduces **probabilistic Conformance Checking**, a new notion of probabilistic compliance for probabilistic traces against declarative process specifications expressed as DECLARE [2] constraints. The DECLARE language models processes through constraints, rules governing the activities within a trace with qualitative temporal relations among the activities. For instance, the constraint `response(a, b)` mandates that each instance of activity `a` must be eventually followed by `b`.

Compliance is computed as the sum of probabilities of all possible "worlds" (deterministic traces derived from the probabilistic trace) that satisfy the constraints. The approach is implemented using an Abductive Logic Programming (ALP) framework [3] and tested on a real-world healthcare protocol (ERAS® guidelines). Experiments demonstrate scalability, handling up to 18 probabilistic events and 21 constraints, though with exponential time complexity in worst-case scenarios.

2 Relevance of the paper to Logic Programming

The paper's contributions are deeply rooted in Probabilistic Logic Programming (PLP). The core innovation lies in adapting the Distribution Semantics [4]—a cornerstone of PLP languages like LPADs, ProbLog, and CP-logic—to process mining:

- In PLP, a probabilistic program defines a distribution over "worlds" (deterministic logic programs). Each world is a possible realization of the program's clauses. The paper mirrors this

by treating a probabilistic trace as a generator of worlds (deterministic traces), where each world includes/excludes events based on their probabilities.

Example: A trace with two probabilistic events $\langle 0.7 : a, 0.5 : b \rangle$ yields four worlds, analogous to how an LPAD with two probabilistic clauses generates four possible worlds.

- The compliance probability is computed by summing probabilities of compliant worlds (traces) to a declarative process specification, paralleling PLP's marginalization over worlds where a query holds.
- The paper leverages the SCIFF framework, an ALP system, to operationalize conformance checking: DECLARE constraints are mapped to integrity constraints (ICs) in SCIFF. In [3], moreover, the authors extended SCIFF with IFF^{Prob} enabling non ground abduction over probabilistic events, which allows us to map probabilistic events in process traces to probabilistic ICs.

3 Significance of the results

- Theoretical advancements: a novel semantics for formalization of probabilistic traces in declarative PM. The approach subsumes deterministic traces (events' probabilities = 1).
- Practical applications: handling of unreliable real-world logs (in our case we demonstrated the application in an healthcare scenario).
- Methodological innovation: combination of probabilistic, abductive, and reasoning about compliance in PM.

References

[1] Michela Vespa, Elena Bellodi, Federico Chesani, Daniela Loreti, Paola Mello, Evelina Lamma & Anna Ciampolini (2024): *Probabilistic compliance in declarative process*. In Giuseppe De Giacomo, Valeria Fionda, Fabiana Fournier, Antonio Ielo, Lior Limonad & Marco Montali, editors: *Proceedings of the 3rd International Workshop on Process Management in the AI Era (PMAI 2024), co-located with 27th European Conference on Artificial Intelligence (ECAI 2024), Santiago de Compostela, Spain, October 19, 2024*, *CEUR Workshop Proceedings* 3779, CEUR-WS.org, pp. 11–22. Available at `https://ceur-ws.org/Vol-3779/paper1.pdf`.

[2] M. Pesic (2008): *Constraint-based workflow management systems: shifting control to users*. Phd thesis 1 (research tu/e / graduation tu/e), Industrial Engineering and Innovation Sciences, doi:10.6100/IR638413. Proefschrift.

[3] Elena Bellodi, Marco Gavanelli, Riccardo Zese, Evelina Lamma & Fabrizio Riguzzi (2021): *Nonground Abductive Logic Programming with Probabilistic Integrity Constraints*. *Theory and Practice of Logic Programming* 21(5), pp. 557–574, doi:10.1017/S1471068421000417.

[4] Taisuke Sato (1995): *A Statistical Learning Method for Logic Programs with Distribution Semantics*. In Leon Sterling, editor: *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995*, MIT Press, pp. 715–729, doi:10.7551/mitpress/4298.003.0069.

Discovery of Logic-probabilistic Rules from COVID-19 Vaccine Antibody Response in Older People: Results from the GeroCovid VAX Study

Michela Vespa
Department of Engineering
University of Ferrara, Italy
michela.vespa@unife.it

Francesca Remelli
Department of Medical Sciences
University of Ferrara, Italy
francesca.remelli@unife.it

Elena Bellodi
Department of Engineering
University of Ferrara, Italy
elena.bellodi@unife.it

Raffaele Antonelli Incalzi
Università Campus Bio-Medico di Roma
Italy
r.antonelli@policlinicocampus.it

Michela Vespa, Francesca Remelli, Elena Bellodi, Raffaele Antonelli Incalzi. Discovery of logic-probabilistic rules from COVID-19 vaccine antibody response in older people: results from the GeroCovid VAX Study. Artificial Intelligence in Medicine - 23nd International Conference, AIME 2025, Pavia, Italy, June 23-25 2025. Pages 1-10. Lecture Notes in Artificial Intelligence (LNAI), Springer. *In press.*

URL https://github.com/bovaristEliza/AIME2025

Keywords Probabilistic Logic Programming, Integrity constraints, Biomedical Data Mining, COVID-19 vaccine

Statement The work the submission is based on has not been already presented at a major logic programming forum.

M. Gebser, D. Inclezan, F. Ricca, M. Carro & M. Truszczynski:
41st International Conference on Logic Programming (ICLP 2025)
EPTCS 439, 2026, pp. 552–554, doi:10.4204/EPTCS.439.40

© Vespa, Remelli, Bellodi & Antonelli Incalzi
This work is licensed under the
Creative Commons Attribution License.

1 Extended Abstract

The SARS-CoV-2 pandemic disproportionately impacted older adults, underscoring significant variability in vaccine-induced immune responses driven by a complex interplay of age-related physiological changes, multimorbidity, polypharmacy, and frailty. These characteristics, prominent in geriatric patient populations such as those studied in the GeroCovid VAX project [3], lead to clinical datasets characterized by high levels of noise, heterogeneity, and incomplete information. Consequently, traditional statistical approaches and classical symbolic Artificial Intelligence (AI) methods often fall short in accurately modeling and interpreting such data. In this context, it is essential to adopt methods capable of explicitly handling uncertainty and maintaining interpretability for clinical decision-making.

To address this, we relied on Probabilistic Logic Programming (PLP), combining the representational capabilities of logic programming with probabilistic reasoning.

This paper leverages PASCAL [1], an algorithm designed explicitly for learning Probabilistic Constraint Logic Theories (PCLTs) from datasets with positive and negative examples. By applying PASCAL to the GeroCovid Vax clinical records, we demonstrate that PLP methods significantly enhance the ability to extract meaningful and interpretable probabilistic rules from data characterized by uncertainty and incompleteness.

2 Main Contributions of the Paper

This work contributes significantly in the following key aspects:

1. **Discovery of meaningful rules for the GeroCovidVax Study:** We used PASCAL to learn clinically interpretable probabilistic rules from the subsample group of the study recording the immune response at timed intervals from the first dose of the Covid-19 vaccine. We were able to discover rules both new and confirmed by literature, declarative probabilistic integrity constraints about factors affecting immune response in elderly populations post-COVID-19 vaccination, offering interpretable support to medical staff.
2. **Superior performance w.r.t. ILP systems:** Comparative evaluation against traditional ILP systems (Aleph, TILDE) and Association Rule Mining (ARM) approaches (FP-Growth, Eclat) reveals that PASCAL significantly outperforms them under conditions of severe class imbalance and data noise.

3 Relevance of the Paper to Logic Programming

PASCAL learns theories composed of Probabilistic Integrity Constraints (PICs) of the form $p :: Body \rightarrow Head$, where $p \in [0,1]$ is the probability, and $Body, Head$ are (dis)junctions of literals. Under the Distribution Semantics [2], these probabilistic constraints independently define a distribution over deterministic theories, thereby capturing uncertainty in real-world scenarios. The learning problem in PASCAL is twofold. Structure learning involves identifying a set of syntactically valid constraints from a hypothesis space defined by a declarative language bias. Parameter learning aims to estimate the optimal weights p for each induced constraint. The probability is to be interpreted as the strength of the IC: a value p means that the sum of the probabilities of the possible theories where a grounding of the constraint is present is p.

This work thus contributes to relevant **real-world applications of (probabilistic) logic programming**.

4 Experimental Evaluation and Significance of the Results

Dataset and Setup. Our experiments utilized clinical data from 611 long-term care residents enrolled in the GeroCovid VAX study. This dataset included antibody response measurements collected at multiple time points (0, 2, 6, 12 months post-vaccination) and detailed patient comorbidity profiles. Each patient's record was represented as a set of ground FOL facts (e.g., `antibody_response_6(it_0044_003650,5.25)`,`frailty(it_0044_003650)`). Supervised labels were derived from antibody response sufficiency ($\geq$33.8 BAU/ml) and mortality outcomes.

Results and Discussion. The evaluation confirmed the superior capability of PASCAL in handling severe class imbalance and real-world data incompleteness (many patients had missing data). PASCAL reached an AUC-ROC value of 1 in mortality prediction tasks, significantly surpassing results from comparative ILP and ARM methods that yielded near-random performance.

Detailed analysis of learned probabilistic constraints highlighted clinically significant relationships, such as: (i) probabilistic associations between inflammatory bowel disease, chronic kidney disease, and reduced vaccine-induced antibody responses, (ii) identification of thyroid disorders as probabilistically predictive of lower antibody responses at 12 months after the first dose, (iii) recognition of combined effects of low antibody levels at 6 months and extensive polypharmacy on elevated mortality risk, aligning closely with existing geriatric medical literature.

References

[1] Fabrizio Riguzzi, Elena Bellodi, Riccardo Zese, Marco Alberti, and Evelina Lamma. Probabilistic inductive constraint logic. *Machine Learning*, 110:723–754, 2021, doi:10.1007/s10994-020-05911-6.

[2] Taisuke Sato. A statistical learning method for logic programs with distribution semantics. In *Proc. of the 12th Int. Conf. on Logic Programming (ICLP'95)*, pages 715–729. MIT Press, 1995, doi:10.7551/mitpress/4298.003.0069.

[3] Caterina Trevisan, Susanna Del Signore, Stefano Fumagalli, Pietro Gareri, Alba Malara, Enrico Mossello, Stefano Volpato, Fabio Monzani, Alessandra Coin, Giuseppe Bellelli, Gianluca Zia, Anette Hylen Ranhoff, and Raffaele Antonelli Incalzi. Assessing the impact of COVID-19 on the health of geriatric patients: The European GeroCovid observational study. *European Journal of Internal Medicine*, 87:29–35, 2021, doi:10.1016/j.ejim.2021.01.017.