EPTCS 416

Proceedings of the

40th International Conference on Logic Programming

University of Texas at Dallas, Dallas Texas, USA, October 14-17 2024

Edited by: Pedro Cabalar, Francesco Fabiano, Martin Gebser, Gopal Gupta
and Theresa Swift

Published: 13th February 2025
DOI: 10.4204/EPTCS.416
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Introduction to the Proceedings of the 40th International Conference on Logic Programming

Pedro Cabalar
University of A Coruña, Spain
pedro.cabalar@udc.es

Theresa Swift
Johns Hopkins Applied Physics Lab
theresasturn@gmail.com

Francesco Fabiano
New Mexico State University
ffabiano@nmsu.edu

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

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

The 40th International Conference on Logic Programming (ICLP 24), was held in Dallas, Texas on October 14-17, 2024, and was co-located with the *17th International Conference on Logic Programming and Non-monotonic Reasoning* held on October 11-14, 2024. This volume contains Technical Communications in Section 1, papers from the affiliated Doctoral Consortium in Section 2, and in Section 3 abstracts from the ICLP 24 invited talks and tutorials.[1]

1 Technical Communications

ICLP 24 technical communications include several types of contributions: regular papers, short papers, and extended abstracts of regular papers. Under the rubric of technical communications we also include extended abstracts of system demos and of recently published research. The high quality of all contributions has been ensured by triple-revieweing; and apart from extended abstracts of recently published research, all contributions in this volume are original work.

Technical communications often represent research that is very new and sometimes longer versions of extended abstracts are published in other venues. However, the impact of technical communications can be quite high. In fact, the *Alain Colmerauer 10-year Test of Time* Award for ICLP 2024 was given to a technical communication, "*Clingo = ASP + Control: Preliminary Report*"[2], by Martin Gebser, Roland Kaminski, Benjamin Kaufmann and Torsten Schaub, originally published as part of the *Proceedings of the Thirtieth International Conference on Logic Programming 2014*.

We loosely group the technical communications in this volume as follows.

Logic Programming and Neural Models Given the phenominal capabilities of recent Large Language Models (LLMs) such as Chat-GPT and Llama, a number of technical communications in ICLP 24 explored how logic programming can be combined with LLMs in meaningful ways, or included within neuro-symbolic approaches.

- *On LLM-generated Logic Programs and their Inference Execution Methods* by Paul Tarau
- *Visual Graph Question Answering with ASP and LLMs for Language Parsing* by Jakob Johannes Bauer, Thomas Eiter, Nelson Higuera Ruiz and Johannes Oetsch

[1] Selected papers from ICLP 24 will be published separately in *Special Issue on the 4oth International Conference of Logic Programming*, *Theory and Practice of Logic Programming* (2025).

[2] (CoRR, abs/1405.3694 (2014)

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.): 40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. iv–xii, doi:10.4204/EPTCS.416.0

- *LLM+Reasoning+Planning for Supporting Incomplete User Queries in Presence of APIs* by Sudhir Agarwal, Anu Sreepathy, David H. Alonso and Prarit Lamba
- *Logical Lease Litigation Prolog and LLMs for Rental Law Compliance in New York* by Sanskar Sehgal and Yanhong A. Liu
- *LP-LM: No Hallucinations in Question Answering with Logic Programming* by Katherine Wu and Yanhong A. Liu.
- *Neuro-symbolic Contrastive Learning for Cross-domain Inference* by Mingyue Liu, Ryo Ueda, Zhen Wan, Katsumi Inoue and Chris Willcocks

Autonomy and Agents Another active research topic investigates the role of logic programming in autonomous, distributed and adaptive dynamic systems.

- *Architecture for Simulating Behavior Mode Changes in Norm-Aware Autonomous Agents* by Sean Glaze and Daniela Inclezan
- *Policies, Penalties, and Autonomous Agents (Extended Abstract)* by Esteban Guerrero and Juan Carlos Nieves
- *Mind the Gaps: Logical English, Prolog, and Multi-agent Systems for Autonomous Vehicles* by Galileo Sartor, Adam Wyner and Giuseppe Contissa
- *Simulating Supply-Chain Contract Execution: A Multi-Agent Approach (Extended Abstract)* by Long Tran, Tran Cao Son, Dylan Flynn and Marcello Balduccini
- *Modular Stochastic Rewritable Petri Nets* by Lorenzo Capra

Explanation and Argumentation An important advantage of logic compared to neural models is that logical reasoning can be explained in human terms using different techniques, including causality and argumentation. This was the topic of three papers in ICLP 24.

- *Semantic Argumentation using Rewriting Systems (Extended Abstract)* by Vineel Tummala and Daniela Inclezan
- *Data2Concept2Text: An Explainable Multilingual Framework for Data Analysis Narration* by Flavio Bertini, Alessandro Dal Palù, Federica Zaglio, Francesco Fabiano and Andrea Formisano
- *Counterfactual Explanations as Plans* by Vaishak Belle[3]

Answer Set Programming As with other recent ICLPs, the topic of Answer Set Programming (ASP) was well-represented. This year, the ASP topics for technical communications included the abductive capabilities of ASP for Visual Query Answering (VQA), the regular models of an ASP program, the use of ASP for meta-reasoning with ontologies, the relations among epistemic extensions of ASP, and using ASP to drive interfaces to other ASP systems.

- *Abduction of Domain Relationships from Data for VQA* by Al Mehdi Saadat Chowdhury, Paulo Shakarian and Gerardo I. Simari
- *Graphical Conditions for Existence, Unicity and Multiplicity of Non-Trivial Regular Rodels* by Van-Giang Trinh, Belaid Benhamou, Sylvain Soliman and Francois Fages

[3]This paper was actually presented in ICLP 23 but inadvertantly omitted from the Proceedings, being eventually published now in this ICLP 24 volume.

- *Efficient OWL2QL Meta-reasoning Using ASP-based Hybrid Knowledge Bases* by Haya Majid Qureshi and Wolfgang Faber
- *Pearce's Characterisation in an Epistemic Domain (Extended Abstract)* by Ezgi Iraz Su
- *ASP-driven User-interaction with Clinguin* by Alexander Beiser, Susana Hahn and Torsten Schaub

Prolog Types, and Unification New perspectives on types and unification algorithms played a part in several technical communications. These papers concerned Prolog as well as other logic-oriented systems and frameworks, including the Coq proof assistant.

- *A Prolog Program for Bottom-up Evaluation* by David S. Warren
- *Regular Typed Unification* by João Barbosa, Mário Florido and Vitor Santos Costa
- *Order-Sorted Intensional Logic: Expressing Subtyping Polymorphism with Typing Assertions and Quantification over Concepts* by Djordje Marković and Marc Denecker
- *A Coq Formalization of Unification Modulo Exclusive-Or* by Yichi Xu, Daniel J. Dougherty and Rose Bohrer

Applications Several technical communications described applications that were developed using annotation logic, abduction, constraint programming, and ASP.

- *Geospatial Trajectory Generation via Efficient Abduction: Deployment for Independent Testing* by Divyagna Bavikadi, Dyuman Aditya, Devendra Parkar, Paulo Shakarian, Graham Mueller, Chad Parvis and Gerardo I. Simari
- *Towards Mass Spectrum Analysis with ASP (Extended Abstract)* by Nils Küchenmeister, Alex Ivliev and Markus Krötzsch
- *Monitoring and Scheduling of Semiconductor Failure Analysis Labs (Extended Abstract)* by Elena Mastria, Domenico Pagliaro, Francesco Calimeri, Simona Perri, Martin Pleschberger and Konstantin Schekotihin

Recently Published Research and Demos

- *Declarative AI design in Unity using Answer Set Programming* by Denise Angilica, Giovambattista Ianni, Francesco Pacenza and Jessica Zangari. (Extended Abstract of recently published research.)
- *stableKanren: Integrating Stable Model Semantics with miniKanren* by Xiangyu Guo, James Smith and Ajay Bansal. (Extended Abstract of recently published research.)
- *Alda: Integrating Logic Rules with Everything Else, Seamlessly* by Yanhong A. Liu, Scott Stoller, Yi Tong and Bo Lin. (System Demonstration.)

2 Doctoral Consortium Papers

The Doctoral Consortium of ICLP 24 was jointly organized with the 17th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR 24). Applications were submitted by 12 PhD students from universities in the US (5), Germany (2), Belgium, Canada, Czechia, Italy and Singapore. Their research summaries received three reviews each by senior members of the ICLP 24 and LPNMR 24 research communities, giving critical yet constructive feedback on the student contributions.

- *Generating Causally Compliant Counterfactual Explanations using ASP* by Sopam Dasgupta
- *Bridging Deep Learning and Logic Programming for Explainability through ILP* by Talissa Dreossi
- *Computational methods for Dynamic Answer Set Programming* by Susana Hahn
- *Relating Answer Set Programming and Many-sorted First-order Logic* by Zachary Hansen
- *Answer Set Counting and its Applications* by Mohimenul Kabir
- *Logical Foundations of Smart Contracts* by Kalonji Kalala
- *Commonsense Reasoning-Aided Autonomous Vehicle Systems* by Keegan Kimbrell
- *A Category-Theoretic Perspective on Approximation Fixpoint Theory* by Samuele Pollaci
- *Hybrid Answer Set Programming: Foundations and Applications* by Nicolas Rühling
- *Autonomous Task Completion Based on Goal-directed Answer Set Programming* by Alexis Tudor
- *Early Validation of High-level Requirements on Cyber-Physical Systems* by Ondřej Vašíček
- *Reliable Conversational Agents under ASP Control that Understand Natural Language* by Yankai Zeng

The Autumn School on Logic Programming, held in conjunction with the Doctoral Consortium, featured four tutorials presented by the following senior researchers.

- Tran Cao Son, New Mexico State University. Las Cruces, NM.
- Y. Annie Liu, Stony Brook University. Stony Brook, NY.
- Manuel Hermenegildo, Technical University of Madrid. Madrid, Spain.
- Torsten Schaub and Susana Hahn, University of Potsdam. Potsdam, Germany.

These Autumn School presenters also shared their experiences and ideas with the PhD students during a mentoring lunch event on October 13.

3 Abstracts of Invited Talks and Tutorials

ICLP 24 had four invited talks and two invited tutorials.

Invited Talk

Logic Programming and Logical Algorithmics
Moshe Vardi
Rice University. Houston, Tx.

Moshe Vardi presented his talk in a plenary session attended by ICLP 24 particiants together with participants of the *17th International Conference on Logic Programming and Non-monotonic Reasoning.*

Abstract Logic programming, born around 1972, expresses a program as a set of Horn clauses. Computation is then performed by applying logical reasoning to that set of clauses. The approach was eventually described as ”Algorithm=Logic+Control”. Another approach to logic and algorithms was developed by Codd in the early 1970s. In his approach, the problem domain is expressed as a relational database, and the problem is expressed as a first-order formula, called “query”. Computation is performed by a meta-algorithm, the query-evaluation algorithm. In this talk, I will describe this approach, which I call Logical Algorithmics, in detail. I will show how this approach yielded multi-variate computational-complexity theory, which offers a more nuanced approach to complexity analysis. It also enabled the development the model-checking algorithms, which are today used in industrial semiconductor design tools.

Invited Tutorial

Logic rules and commonsense in uncertain times:
A simple unified semantics for reasoning with assurance and agreement

Y. Annie Liu
Stony Brook University. Stony Brook, NY.

Abstract Complex reasoning problems are most clearly and easily specified using logical rules, but require recursive rules with aggregation such as *count* and *sum* and more for practical applications. Unfortunately, the meaning of such rules has been a significant challenge, with many disagreeing semantics, baffling commonsense for rigorous logic.

This tutorial examines a simple unified semantics for reasoning with assurance and agreement, and consists of three main parts:

1. An introduction to complex reasoning problems expressed using logic rules, with recursion, negation, quantification, and aggregation; the key idea of a simple unified semantics, supporting simple expression of different assumptions; and how it unifies different prior semantics.
2. An overview of the precise rule language; the formal semantics, called Founded Semantics and Constraint Semantics, or Founded+Constraint Semantics (FCS) for short here, supporting efficient and precise inference over aggregation even with approximation; and the properties of the semantics.
3. An exploration of a wide range of challenging examples, including the well-known problem of company control and extended win-not-win games. FCS is simple and matches the desired results in all cases.

Additionally, we discuss how to combine logic/rigorous languages and LLMs/ML for problem solving and question answering with assurance, where a simple unified semantics is critical for its generality, power, and ease of use.

Invited Talk

Linear Algebraic Approaches to Logic Programming

Katsumi Inoue
National Institute of Informatics. Chiyoda, Japan.

Abstract Integration of symbolic reasoning and machine learning is important for robust AI. Realization of symbolic reasoning based on algebraic methods is promising to bridge between symbolic reasoning and machine learning, since algebraic data structures have been used in machine learning.

To this end, Sakama, Inoue and Sato have defined notable relations between logic programming and linear algebra and have proposed algorithms to compute logic programs numerically using tensors. This work has been extended in various ways, to compute supported and stable models of normal logic programs, to enhance the efficiency of computation using sparse methods, and to enable abduction for abductive logic programming. A common principle in this approach is to formulate logical formulas as vectors/matrices/tensors, and linear algebraic operations are applied on these elements for computation of logic programming. Partial evaluation can be realized in parallel and by self-multiplication, showing the potential for exponential speedup.

Furthermore, the idea to represent logic programs as tensors and matrices and to transform logical reasoning to numeric computation can be the basis of the differentiable methods for learning logic programs.

Invited Talk

The Anatomy of the SICStus Finite-Domain Constraint Solver

Mats Carlsson
Research Institute of Sweden. Kista, Sweden.

Abstract Constraint programming (CP) is a powerful problem-solving paradigm with roots in combinatorics, linear programming, logic programming, and AI. A notable development in the 1980s was the fusion of constraint solving with logic programming into constraint logic programming (CLP), which extended Prolog by integrating constraints into its framework. To extend a Prolog system with constraints, a large number of algorithms and data structures must be added for tasks like domain representation, constraint propagation, search, a whole menagerie of filtering algorithms for specific constraints, and peaceful coexistence with the Prolog virtual machine and runtime system.

This talk focuses on the constraint programming support in SICStus Prolog: the key extensions to Prolog that were necessary, details of the solver architecture, and a discussion of design choices. I will try to put the work in a historical perspective and also say something about programming interfaces, use cases, and my outlook on the future of CP.

Invited Tutotial

Encoding High-Level Constraints into SAT and MIP

Neng-Fa Zhou
CUNY Brooklyn College and Graduate Center. Brooklyn, NY.

Abstract: Picat provides four solver modules, including CP, SAT, MIP, and SMT, for modeling and solving constraint satisfaction and optimization problems (CSPs). This tutorial introduces the inner workings of Picat's SAT and MIP modules. PicatSAT encodes constraints into SAT based on unary and binary encodings of domain variables. PicatSAT adopts many optimizations from CP systems, language compilers, and hardware design systems for encoding primitive constraints into compact and efficient SAT encodings. PicatSAT also employs some novel algorithms for decomposing global constraints, especially graph constraints. PicatMIP, while generally not as competitive as PicatSAT on MiniZinc and

XCSP benchmarks, is a good supplementary solver for CSPs. PicatMIP adopts the well-known big-M method for linearizing nonlinear constraints, and employs some optimizations for translating special nonlinear constraints into linear ones.

Invited Talk

How Structure Shapes Logic Programming and Counting-Based Reasoning

Markus Hecher
Massachusetts Institute of Technology. Cambridge, Ma.

Abstract When can we efficiently solve combinatorially hard problems? In practice, state-of-the-art solvers can tackle instances with millions of variables, creating a significant gap between empirical performance and theoretical limits. A key factor in bridging this gap is understanding the structural properties of problem instances. In this talk, we explore how to efficiently leverage these structures, with a particular focus on the role of treewidth and the answer set programming framework. We establish tight runtime upper and lower bounds, grounded in reasonable complexity-theoretic assumptions. Special attention is given to counting-based reasoning, a computationally intensive task where structure plays a critical role. Through empirical results, we demonstrate how structural insights can drastically improve the efficiency of counting in combinatorial problem-solving. This emphasizes the importance of theoretical studies and their practical applications, showcasing how we bring theory and practice closer together.

Acknowledgements

We would like to begin by thanking everyone who together organized ICLP 24 and contributed to its success. These include

- General Chair: Gopal Gupta, University of Texas at Dallas
- Honorary General Chair: Doug DeGroot, University of Texas at Dallas
- Workshop Chair: Joaquín Arias, Universidad Rey Juan Carlos
- Webmasters: Abhiramon Rajasekharan, University of Texas at Dallas; Keegan Kimbrell, University of Texas at Dallas; and Varad Abhyankar, University of Texas at Dallas.
- Local Organizing Committee: Varad Abhyankar, Sopam Dasgupta. Doug DeGroot. Serdar Erbatur. Keegan Kimbrell. Richard Min, Parth Padalkar, Abhiramon Rajasekharan, Elmer Salazar, Jey Veerasamy, Yankai Zeng.

The ICLP 24 program committee consisted of representatives of logic programming groups in Australia, Austria, Canada, China, the Czech Republic, Finland, Germany, Greece, Israel, Italy, Japan, Netherlands, Portugal, Spain, Sweden, Turkey, the United Kingdom, and the United States. The issue editors feel fortunate for the close involvement of a responsive and energetic program committee. The program committee members were as follows.

Salvador Abreu
Mario Alviano
Nicos Angelopoulos
Joaquín Arias
Marcello Balduccini
Mutsunori Banbara
Chitta Baral
Roman Barták
Elena Bellodi
Bart Bogaerts
Roberta Calegari
Francesco Calimeri
Manuel Carro
Angelos Charalambidis
Michael Codish
Stefania Costantini
Alessandro Dal Palù
Marina De Vos
Marc Denecker
Agostino Dovier
Thomas Eiter
Esra Erdem
Wolfgang Faber
Jorge Fandinno
Johannes K. Fichte
Fabio Fioravanti
Andrea Formisano
Gerhard Friedrich
Marco Gavanelli
Martin Gebser
Laura Giordano
Ricardo Gonçalves
Gopal Gupta
Markus Hecher
Giovambattista Ianni
Daniela Inclezan
Tomi Janhunen
Matthias Kno
Joao Leite
Michael Leuschel
Yuliya Lierler
Vladimir Lifschitz
Francesca Alessandra Lisi
Yanhong A. Liu
Marco Maratea
Viviana Mascardi
Laurent Michel
Jose F. Morales
Johannes Oetsch
Manuel Ojeda-Aciego
Simona Perri
Enrico Pontelli
Francesco Ricca
Fabrizio Riguzzi
Ricardo Rocha
Chiaki Sakama
Vitor Santos-Costa
Zeynep G. Saribatur
Torsten Schaub
Konstantin Schekotihin
Tom Schrijvers
Tran Cao Son
Mohan Sridharan
Paul Tarau
Hans Tompits
Mirek Truszczynski
German Vidal
Concepcion Vidal
Alicia Villanueva
Kewen Wang
David Warren
Felix Weitkämper
Jan Wielemaker
Stefan Woltran
Roland Yap
Jia-Huai You
Zhizheng Zhang
Yuanlin Zhang
Neng-Fa Zhou

We also thank our external reviewers:

João Barbosa
Manuel Borroto
Geoffrey Churchill
Michele Collevati
Daniele Theseider Dupre
Müge Fidan
Tobias Geibinger
Michael Gelfond
Kenji Hashimoto
Antonio Ielo
Ryan Kepler
Matthias König
Elena Mastria
Ciaran McCreesh
Samuele Pollaci
Maurizio Proietti
Abhiramon Rajasekharan
Javier Romero
Takehide Soh
Gioacchino Sterlicchio
Robbe Van den Eede
Linde Vanbesien
Huaduo Wang.

We are grateful to the University of Texas at Dallas on whose campus the conference was organized. The following organizations within the University provided direct financial support: The Department of Computer Science, The Erik Jonsson School of Engineering and Computer Science, The Office of the Vice President for Research, and The Center for Applied AI and Machine Learning

Additionally, ICLP 2024 received funds for student support from the Artificial Intelligence Journal and the National Science Foundation.

On LLM-generated Logic Programs and their Inference Execution Methods

Paul Tarau[0000−0001−7192−9421]
University of North Texas
paul.tarau@unt.edu

Large Language Models (LLMs) trained on petabytes of data are highly compressed repositories of a significant proportion of the knowledge accumulated and distilled so far. In this paper we study techniques to elicit this knowledge in the form of several classes of logic programs, including propositional Horn clauses, Dual Horn clauses, relational triplets and Definite Clause Grammars. Exposing this knowledge as logic programs enables sound reasoning methods that can verify alignment of LLM outputs to their intended uses and extend their inference capabilities. We study new execution methods for the generated programs, including soft-unification of abducible facts against LLM-generated content stored in a vector database as well as GPU-based acceleration of minimal model computation that supports inference with large LLM-generated programs.

Keywords: LLM-generated logic programs; LLM-generated Definite Clause Grammars; LLM-generated relation graphs; soft-unification with abducible facts; GPU-supported evaluation of propositional Horn clause programs; visualization of LLM-generated relations.

1 Introduction

While the multi-step dialog model initiated by ChatGPT is now available from a few dozen online or locally run open source and closed source LLMs, it does not cover the need to efficiently extract salient information from an LLMs "parameter-memory" that encapsulates in a heavily compressed form the result of training the model on trillions of documents and multimodal data.

Steps in this direction have been taken, relying on ground-truth involving additional information sources (e.g., collections of reference documents or use of web search queries). Among them, we mention work on improving performance of Retrieval Augmented Generation (RAG) systems [7] by recursively embedding, clustering, and summarizing chunks of text for better hierarchical LLM-assisted summarization [15], multi-agent hybrid LLM and local computation aggregators [3] and deductive verifiers of chain of thoughts reasoning [9].

A more direct approach is recursion on LLM queries, by chaining the LLM's distilled output as input to a next step and casting its content and interrelations in the form of logic programs, to automate and focus this information extraction with minimal human input [18, 20]. Like in the case of typical RAG architectures [7, 15], this process can rely on external ground truth but it can also use new LLM client instances as "oracles" deciding the validity of the synthesized rules or facts.

With focus on automation of this unmediated salient knowledge extraction from the LLM's parameter memory and its encapsulation in the form of synthesized logic programming code, we will extend in this paper the work initiated in [18, 20] with:

- new LLM input-output chaining mechanisms
- new types of generated logic programs
- new relational representations elicited from LLM output steps

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 1–14, doi:10.4204/EPTCS.416.1

- scalable execution mechanisms that accommodate very large logic programs at deeper recursion levels
- soft-unification based execution of LLM-generated logic programs as a principled encapsulation of the RAG retrieval process

The rest of the paper is organized as follows. Section 2 overviews the DeepLLM architecture described in [20] and its new extensions supporting the results in this paper. Section 3 overviews the generation of Horn clause programs with the online DeepLLM app. Section 4 explains the LLM-based generation of Dual Horn clause programs and their uses to explore counterfactual consequences and theory falsification, Section 5 introduces the use of DCG grammars as a representation of LLM-generated answer and follow-up question pairs. Section 6 describes fixpoint and GPU-supported minimal model computation for the generated programs. Section 7 describes relation-extraction and visualization from the minimal models of the LLM-generated propositional programs. Section 8 introduces the soft-unification based encapsulation of the information retrieval against facts extracted from authoritative document collections. Section 9 discusses related work and Section 10 concludes the paper.

2 Recursive exploration of LLM dialog threads

Generative AI, with often human-like language skills is shifting focus from typical search engines to more conversational interactions. Yet, the challenge remains that humans must still process and verify this information, an often tedious task.

Our answer to this, as implemented in the DeepLLM system is to automate the entire process. We start with a simple “initiator goal” and let the LLM dive recursively in its parametric memory and deliver a detailed answer focused on the initiator and the trace of this chain of steps summarized as the short term-memory maintained via its API. This automation also helps to minimize common issues like inaccuracies, made-up information, and biases that are often associated with LLMs.

We refer to [20, 18] for details of implementation of the DeepLLM system, as well as to its open-source code[1] and its online demo[2].

The DeepLLM system’s active components (subclasses of the Agent class) are Interactors, Recursors, and Refiners:

- Interactors manage input prompts and task breakdown
- Recursors handle iterative exploration of subtasks
- Refiners enhance clarity and relevance of LLM responses

To validate its reasoning steps, the system also relies on stored knowledge resources:

- Ground truth facts: sentences collected from online sources or local documents
- Vector store: enabler of “semantic search” via embeddings of sentences

Starting from a succinct prompt (typically a nominal phrase or a short sentence describing the task) an Interactor will call the LLM via its API, driven by a Recursor that analyzes the LLM’s responses and activates new LLM queries as it proceeds to refine the information received up to a given depth.

Refiners are Recursor subclasses that rely on semantic search in an embeddings store containing ground-truth facts as well as on oracles implemented as specialized Interactors that ask the LLM for

[1] `https://github.com/ptarau/recursors/`

[2] `https://deepllm.streamlit.app`

advice on deciding the truth of, or the rating of hypotheses. Besides returning a stream of answers, Recursors and Refiners compile their reasoning steps to a propositional Horn clause program available for inspection by the user or subject for execution and analysis with logic programming tools (in particular, with our model builder – a fast propositional Horn clause theorem prover).

3 Generating propositional Horn clause programs with the DeepLLM app

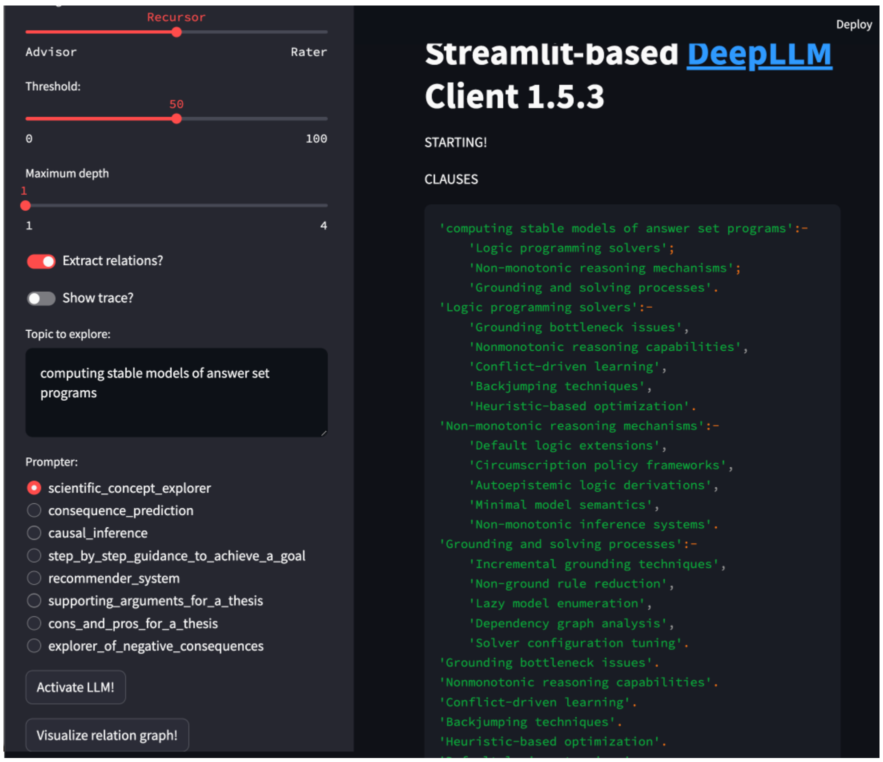


Figure 1: DeepLLM app

We refer to [18] for an extensive list of LLM-generated Horn clause programs. We will just briefly describe here the DeeLLM app (see **Fig. 1**) that we will use for generating our logic programs. In the case of the interaction shown in **Fig. 1**, the initiator goal "computing stable models of answer set programs" starts the "scientific concept explorer" option and generates in the right side window a Horn clause program describing successive refinements of the initiator goal.

The DeeLLM app is written with the `Streamlit`[3] webapp generator and offers the choice between GPT4, GPT3.5 or a local LLM, running as a server and supporting an OpenAI compatible API. It then lets the user choose between the Recursor, Advisor and Rater agents, providing for the latter a threshold level slider. The threshold informs the Rater oracle to accept or reject a generated rule head or fact (the higher the threshold the stricter the accept decision). Options to set the maximum recursion depth and activate relation extraction and visualization are also available.

[3] `https://streamlit.io/`

The application starts once the user enters the topic to explore, chooses the prompter template and activates the LLM. Besides the output produced in the right window, when run locally, it saves the generated logic program and its computed minimal model as Prolog code files.

4 Generating propositional Dual Horn clause programs

A *Dual Horn clause* is a disjunction of literals with at most one negative literal (or exactly one if it is a definite Dual Horn clause). A *Dual Horn clause Program* is a conjunction of Dual Horn clauses. We represent a Dual Horn clause like $\neg p_0 \vee p_1 \vee \ldots \vee p_n$ in an equivalent implicational form $p_0 \rightarrow p_1 \vee \ldots \vee p_n$, similarly to Prolog's representation of Horn clauses. We adopt a Prolog-like syntax, with $\rightarrow$ represented as "`=>`" and $\vee$ represented as ";". Note also that "`s => false`" represents a negated fact the same way as "`s :- true`" would represent a positively stated fact.

The objective of Dual Horn programs is to describe (constructively) why something is not true i.e., to falsify the initiator goal by back-propagating from its negative (or more generally, undesirable, unwanted, harmful, impossible, etc.,) consequences.

For instance, from a clause like `p => q ; r ; s`, assuming that `p` were true, we would infer that at least one of `q` , `r` and `s` should be true. The contrapositive is that if `q` , `r` and `s` are all false, then `p` should be false as well. Like in the case of SLD-resolution on Horn clauses, this triggers a goal oriented process where successful falsification of all consequences results in falsification of the "counterfactual" hypothesis that initiated the process.

By instructing the LLM to infer the negative consequences of the DeepLLM initiator goal, we can obtain a Dual Horn program.

Example 1 *The Dual Horn clauses (recursion level = 0) with heads (starting with consequences of* '`tailgate when driving`'*) in the clause body are:*

```
'tailgate when driving' =>
    'Increased accident risk';
    'Reduced reaction time'.
'Increased accident risk' =>
    'Higher insurance premiums';
    'Severe injury likelihood';
    'Vehicle damage costs';
    'Legal consequences';
    'Emotional trauma impact'.
```

```
'Reduced reaction time' =>
    'Increased accident risk';
    'Delayed braking response';
    'Higher collision likelihood';
    'Compromised driving safety'.
```

The negative facts (unexplored recursion level = 1 goals) are:

```
'Higher insurance premiums' => false.
'Severe injury likelihood' => false.
'Vehicle damage costs' => false.
'Legal consequences' => false.
'Emotional trauma impact '=> false.
```

```
'Delayed braking response' => false.
'Higher collision likelihood' => false.
'Compromised driving safety' => false.
```

Note that "`=> false`" marks things that we do not want to happen, from where the same backpropagates to the initiator `'tailgate when driving'`. *Our compilation algorithm*[4] *will transform this into a definite program placed in module* `false` *that can be queried with:*

```
?- false:'tailgate when driving'.
true
```

Its successful falsification could then advise a car driving program or person to avoid the aforementioned behavior.

A more interesting exploration (at recursion level=2) of negative consequences in the form of a Dual Horn clause program[5] reveals persuasive counter-arguments to unwise political decisions.

Example 2 *A few unwanted consequences at descent level 2 for* `'loosing the FED_s independence'`*:*

```
'loosing the FED_s independence' =>
    'Increased political influence on monetary policy'. % level 0
...
'Increased political influence on monetary policy' =>
    'Politicized interest rates';
    'Short-term economic manipulation';
    'Eroded investor confidence'; % level 1
    'Heightened market volatility';
    'Policy-driven inflation risks'.
 ...
 'Eroded investor confidence' => % level 2
    'Market volatility';
    'Capital flight';
    'Reduced foreign investment'.
```

5 From Self-generated follow-up question-answer chains to DCG grammars

DeepQA[6] (see **Fig. 2**) is a DeepLLM-based application that explores recursively the "mind-stream" of an LLM via a tree of self-generated follow-up questions. Interestingly, by asking the LLM to generate a set of follow-up questions to its own answers creates (especially when the process recurses) a more focused "stream of thoughts", possibly as an emergent property of its "in-context learning" abilities.

After started from an initiator question on a topic of the user's choice, the app explores its tree of *follow-up questions* up to a given depth. As output, it generates a Definite Clause Grammar that can be imported as part of a Prolog program. The DCG, in generation mode, will replicate symbolically the equivalent of the "stream of thoughts" extracted from the LLM interaction, with possible uses of the encapsulated knowledge in Logic Programming applications.

[4] `https://github.com/ptarau/TypesAndProofs/blob/master/symlp/compile_clauses.pro`
[5] full code at `https://github.com/ptarau/output_samples/tree/main/deepllm`
[6] `https://github.com/ptarau/recursors/tree/main/deepQA`

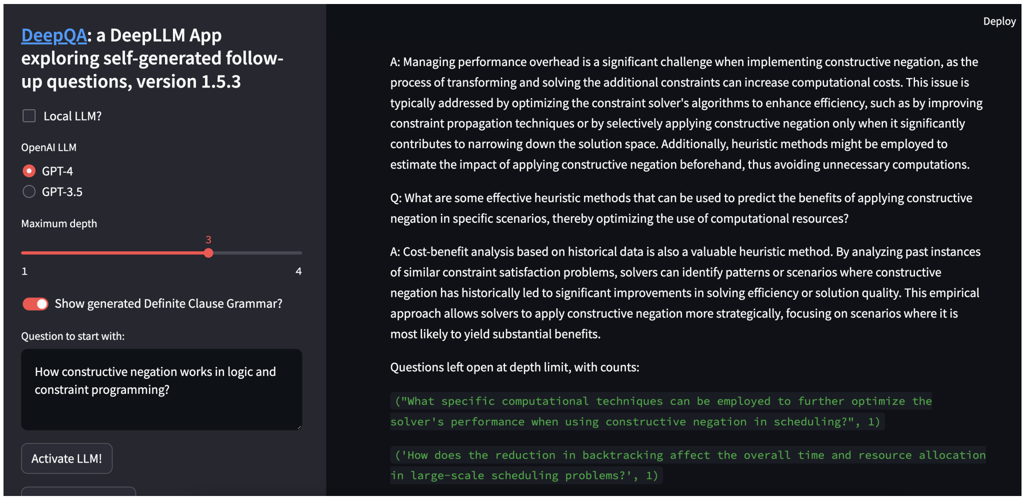


Figure 2: DeepQA with "*How constructive negation works in logic and constraint programming?*"

The synthesized grammar is designed to generate a finite language (by carefully detecting follow-up questions that would induce loops). We also ensure that paths in the question-answer tree are free of repeated answers, which get collected as well, together with questions left open as a result of reaching the user-set depth limit.

Example 3 *Definite Clause Grammar generated by initiator question* 'How constructive negation works in logic and constraint programming?'*:*

```
% DCG GRAMMAR RULES:

q0-->q0_,a0_,q1.
q0-->q0_,a1_,q2.
q1-->q1_,a3_,q4.
...
q12-->q12_,a38_.
```

For instance, the first rule rewrites the initiator `q0` *into:*

- *the terminal* `q0_` *that will produce the actual text of the question*
- *the terminal* `a0_` *that will produce the actual text of the answer to* `q0_`
- *the non-terminal* `q1` *continuing the generation process with one of the follow-up questions generated by the LLM*

```
% QUESTION TERMINALS:

q0_-->['Q: How constructive negation works in logic and constraint programming?'].
q1_-->['Q: Can you provide an example of how constructive negation might refine
   the solution space in a practical constraint programming problem?'].
q2_-->['Q: How does constructive negation differ from classical negation in terms
   of computational efficiency and outcome in constraint satisfaction problems?'].
   ...
```

```
% ANSWER TERMINALS:

a0_-->['A: Constructive negation in logic and constraint programming is a method
    used to handle negation in a way that allows for the derivation of new
    constraints from negative information. Instead of simply rejecting solutions
    that do not satisfy a certain condition, constructive negation works by
    deducing what must be true if a given condition is false. This is particularly
    useful in constraint programming where constraints define what is possible
    rather than what is not. By applying constructive negation, the system can
    infer additional constraints that must be met for the negation to hold,
    effectively refining the solution space.'].
    ...
```

When reaching the user-specified recursion depth, the unanswered follow-up questions are collected as "open questions" in the predicate `opens/2` *with the second argument indicating the number of times (over all branches of the tree) the question has been generated. In the case of an LLM with a very large parametric memory (e.g., GPT4, Claude 3 or Gemini) values above 1 are unlikely, while with smaller LLMs (e.g., Vicuna) repeated follow-up questions can happen more often.*

```
% OPEN QUESTIONS:

opens('What specific computational techniques can be employed to further optimize
     the solver_s performance when using constructive negation in scheduling?',1).
opens('How does the reduction in backtracking affect the overall time and
              resource allocation in large-scale scheduling problems?',1).
...
```

Starting the DCG in generation mode from its `q0` *initiator goal is achieved as follows:*

```
% entry point to generate the language covered by the DCG grammar
go:-q0(Xs,[]),nl,member(X,Xs),write(X),nl,nl,fail.
```

One can also use DeepQA to quickly assess the strength of an LLM before committing to it. For instance, when used with a much weaker than GPT4 local LLM (enabled with Vicuna 7B by default) one will see shorter, more out of focus results, with a lot of repeated questions and answers collected by DeepQA in corresponding bins.

The full Prolog code discussed in thus example is available online[7] as well the DeepQA app[8].

6 Computing minimal models of LLM-generated logic programs

6.1 Minimal model computation with a GPU-friendly Torch-based Linear Algebra Algorithm

At deeper recursion levels, the generated logic programs, providing a symbolic representation of an LLM's parameter memory can quickly reach millions of clauses, ready to reason with.

To take advantage of the significant acceleration provided by GPUs we have implemented a `torch`-based linear algebraic minimal model computation algorithm[9] along the lines of [13].

[7] https://github.com/ptarau/output_samples/tree/main/deepqa
[8] https://deep-auto-quests.streamlit.app/
[9] https://github.com/ptarau/recursors/blob/main/tenslogic/proptens.py

The implementation is centered around

```
def tp(M, v):
    """
    one step fixpoint operator
    """
    r = M @ v
    return (r >= 1.0).to(torch.float32)
```

that advances one step of the fixpoint computation with a matrix multiplication "`@`" and

```
def tp_n(M, v0):
    """
    iterated fixpoint operator
    """
    oldv = v0
    n = M.shape[0]
    for i in range(n):
        newv = tp(M, oldv)
        if torch.allclose(newv, oldv):
            return newv
        oldv = newv
```

that proceeds until a fixpoint is detected using `torch.allclose`.

The program contains readers of Horn clause programs represented in as `.json` files. It can handle medium size programs (a few thousand clauses), as despite the GPU acceleration, complexity is still dominated by $O(N^3)$ matrix products.

We will show here a small test program running the minimal model computation. After defining:

```
top = "true"
bot = "false"

vs = (p, q, r, s) = "pqrs"
```

We represent the program as pair made of the head of the clause and the list of atoms in its body:

```
prog = [
(        p, [q]),
(        p, [r]),
(        q, [r, s]),
(        r, [top]),
(        bot, [q])
]
```

We can then compute the model with:

```
>>> print(compute_model(prog))
['p', 'r']
```

Future work using torch sparse tensors[10], to ensure scalability for very large generated programs is planned along the lines of [11].

[10] `https://pytorch.org/docs/stable/sparse.html`

6.2 Fixpoint-based minimal model computation

It is not unusual to have loops in the propositional Horn Clause program connecting the LLM generated items by our recursors and refiners that would create problems with Prolog's depth-first execution model. As using a SAT-solver would be an overkill in this case, given that Horn Clause and Dual Horn clause formula satisfiability is known to be polynomial, we have implemented a simple low-polynomial complexity [6] propositional satisfiability checker and model builder[11].

The model builder works by propagating truth from facts to rules until a fix point is reached. Given a Horn Clause $h : -b_1, b_2, ..., b_n$, when all b_i are known to be true (i.e., in the model), h is also added to the model. If integrity constraints (Horn clauses of the form $false : -b_1, b_2, ..., b_n$) have also been generated by the oracle agents monitoring our refiners, in the advent that all $b_1, b_2, ..., b_n$ end up in the model, $b_1, b_2, ..., b_n$ implying $false$ signals a contradiction and thus unsatisfiability of the Horn formula associated to the generated program. However as the items generated by our recursive process are not necessarily expressing logically connected facts (e.g., they might be just semantic similarity driven associations), turning on or off this draconian discarding of the model is left as an option for the application developer. Also, the application developer can chose to stop as soon as a proof of the original goal emerges, in a way similar to goal-driven ASP-solvers like [1], irrespectively to unrelated contradictions elsewhere in the program.

7 Generating relation triplets for knowledge graphs

Our DeepLLM app offers an option to generate from the minimal model of the program a relation graph (see **Fig. 3**) consisting of implication links (marked with "`:`") to which it adds generalization links (marked with "`is`").

Implication links are extracted directly from the logic program while generalization links, serving as additional explanations, are generated by the LLM via an additional request.

Several other types of relation graphs can be generated depending on the planned reasoning method.

One of them is extraction of <subject, verb, object> (SVO) triplets obtained by prompting the LLM to split a complex sentence in simpler ones and extract from each simple sentence an SVO triplet.

Another is a hybrid method, combining relations extracted by using dependency grammars [21], embeddings-based similarity relations, Wordnet-based and LLM-generated hypernyms and meronyms.

8 Reasoning with soft unification on noisy facts

The minimal models of LLM-generated Horn clause programs encapsulate facts and their consequences elicited from DeepLLM's initiator queries in the form of natural language sentences. When writing a logic program that performs symbolic reasoning relying on a ground fact database of such sentences, an interesting form of *abductive reasoning* emerges. When hitting an undefined ground sentence, intended as a query to match database facts we can rely on vector embedding of the sentences and proximity of the query and the facts in the vector space as a "good enough" match, provided that the semantic distance between them is below a given threshold. We will next describe a proof of concept of this strategy that we illustrate on a small quotation dataset consisting of a few sentences[12].

[11] `https://github.com/ptarau/recursors/blob/main/deepllm/horn_prover.py`
[12] `https://github.com/ptarau/natlog/blob/main/docs/quotes.txt`

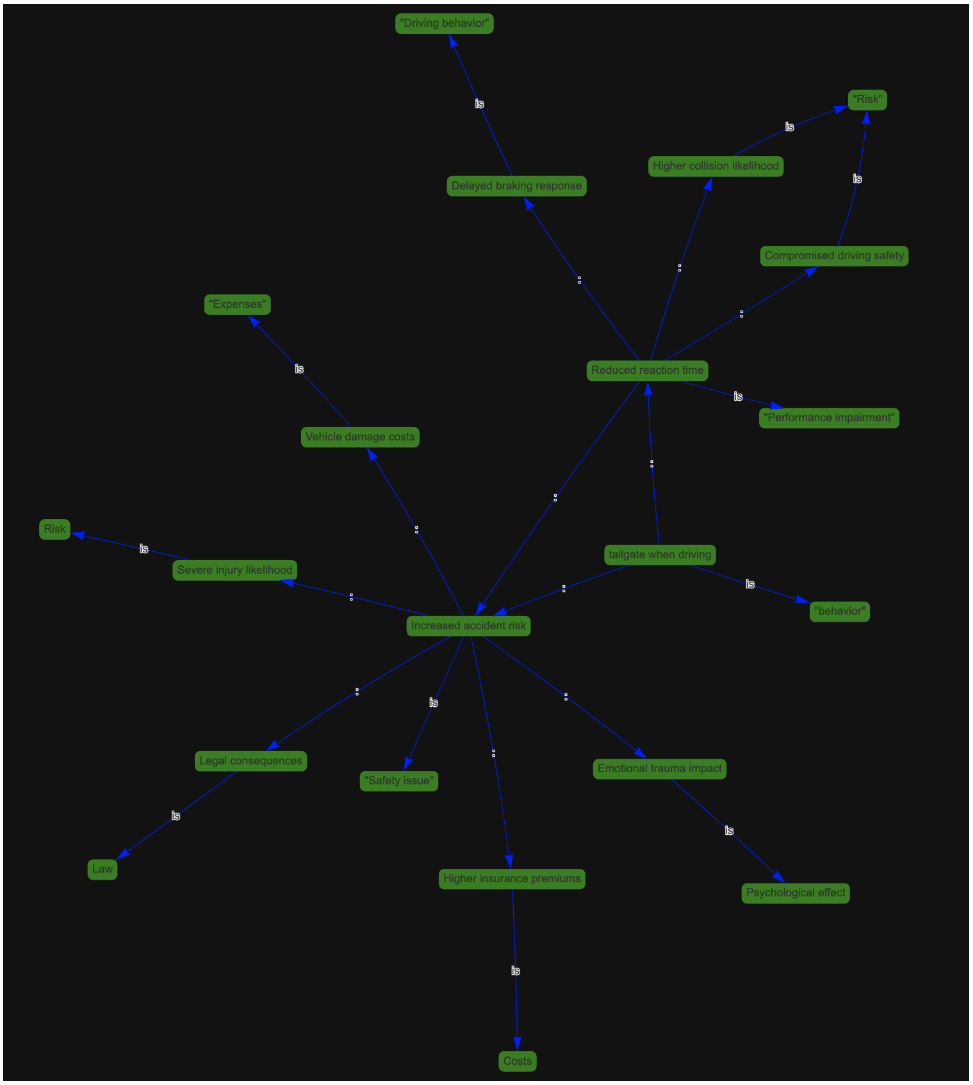


Figure 3: Relation graph for “*tailgate when driving*”

We have implemented `Softlog`[13] as an extension to the `Natlog` system [17, 19], a Python-based Prolog dialect with a simpler syntax and a lightweight Python interface. It works simply by overloading `Natlog`'s built-in ground fact database's unification method with a form of *soft-unification* [4, 2, 10], implemented as follows:

```
    def unify_with_fact(self, goal, trail):

        # q = query goal to be matched
        # k = number of knns to be returned
        # d = minimum knn distance
        # v = variable to be unified with the matches

        q, k, d, v = goal
        d = float(d) / 100
        _, answers = self.emb.knn_query(q, k)
        for sent, dist in answers:
            if dist <= d:
                self.abduced_clauses[(q, sent)] = dist
                yield unify(v, sent, trail)
```

The following `Natlog` script is then used to query a small set of sentences serving as `Softlog`'s ground database. Note that the "~" symbol is Natlog's convention for marking calls to a ground (soft-)database.

```
knn 3.
threshold 70.

quest  Quest Answer:
  knn K, % K is the number passed to the K Nearest Neighbors query
  threshold D,
  ~ Quest K D Answer.
```

We implement soft unification queries as K closest neighbors (KNN) computations against embeddings in our `sentence_store`[14]. We use Sentence Transformers [12] to compute embeddings and store them locally in an efficient and scalable vector database. As usual in `Natlog`, the Python iterator returning multiple KNN matches is mapped to Prolog's backtracking with multiple answers returned as alternative bindings to a result variable.

```
?- quest 'What happens if you do not know where you go' X?
ANSWER: {'X': 'If you don t know where you are going
               you will end up somewhere else said Yogi Berra.'}
ANSWER: {'X': 'If you don t know where you are going
               any road will get you there said Lewis Carroll.'}
```

When a query `(Q A)` binds `A` to an answer extracted from the vector store, a binary clause `(Q :- A)` and its supporting fact `(A :- true)` are inserted into the dictionary of *abduced clauses*. If we add them to the program that triggered the generation of the clauses, we obtain a self-contained standard logic program that returns exactly the same answers as its `Softlog` counterpart. Alternatively, the computed distances can be normalized as probabilities, to annotate clauses used in a Probabilistic Logic Programming language like Problog [5].

[13] `https://github.com/ptarau/natlog/tree/main/softlog`
[14] `https://github.com/ptarau/sentence_store`

```
ABDUCED CLAUSES:

'What happens if you do not know where you go' :
    'If you don t know where you are going you will end up somewhere else
    said Yogi Berra'. % distance=0.5874345302581787
'What happens if you do not know where you go' :
    'If you don t know where you are going any road will get you there
    said Lewis Carroll'. % distance=0.6724047660827637
```

Note that by contrast to the usual exact unification based answers, `Softlog` works quite well when the query is *close enough* to a matching entry in the sentence store, a reasonable assumption when the facts have been generated from multiple LLM runs and several ground truth resources.

```
?- quest 'What did Wilde say about temptation' X?
ANSWER: {'X': 'I can resist anything except temptation said Oscar Wilde.'}

?- quest 'What did Alice say about following advice' X?
ANSWER: {'X': 'I give myself very good advice but I very seldom follow it
               said Lewis Carroll.'}
```

Given the nature of semantic search, surname is enough to find Oscar Wilde and as `Alice` associates with the author Lewis Carroll, soft unification will fetch it from the sentence store.

9 Related Work

By contrast to "neuro-symbolic" AI [14], where the neural architecture is closely intermixed with symbolic steps, in our approach the neural processing is encapsulated in the LLMs and accessed via a declarative, high-level API. This reduces the semantic gap between the neural and symbolic sides as their communication happens at a much higher, fully automated and directly explainable level.

Our recursive descent algorithm shares the goal of extracting more accurate information from the LLM interaction with work on "Chain of Thought" prompting of LLMs [22, 9] and with step by step [8] refinement of the dialog threads. Our approach shares with tools like LangChain [3] the idea of piping together multiple instances of LLMs, computational units, prompt templates and custom agents, except that we fully automate the process without the need to manually stitch together the components.

We have not found any references to the use of Dual Horn clauses in logic programming but it is a well known result [16]) that their complexity in the propositional case is polynomial, similarly to their of Horn clause counterparts. This fact makes them also good generation targets for LLM-extracted knowledge processing.

We have not found anything similar to generating question-answer-follow-up question chains, although it is common practice for chatbots to suggest (a choice between) follow-up questions[15].

Our torch-based model-computation algorithm follows closely the matrix-computation logic of [13], our contribution being its succinct and efficient GPU-friendly implementation.

Interest in several forms of soft-unification has been active [4, 2, 10] as differentiable substitute of symbolic unification in neuro-symbolic systems. By contrast, our focus in this paper is flexible information retrieval of LLM-generated natural language content, for which high quality embeddings were available either from LLM APIs or local resources like the torch-based sentence-transformers [12].

[15] including the author's own `https://auto-quest.streamlit.app/`

10 Conclusion

It is now undeniable that Generative AI is a major disruptor not just of industrial fields ranging from search engines, automation of software development and robotics to medical and legal advisory systems, but also a disruptor of research fields, including symbolic AI as we know it and machine Learning itself. In particular, results produced by dominant ML or NLP techniques as well as work on integration of neural and symbolic systems have become replaceable by much simpler applications centered around LLM queries and RAG systems. In fact, by concentrating the knowledge encapsulated in its parametric memory into a single declarative interface, Generative AI can replace complex, labor-intensive software functionality with a simple LLM API call or a question in one's favorite natural language.

This motivates our effort to "join the disruption" and explore several new ways to elicit the knowledge encapsulated in the LLMs' parametric memory as logic programs, together with an investigation of their optimal inference execution methods. We have not just exposed as logic programs the several kinds of knowledge snippets extracted by recursive automation LLM dialog threads , but we have also devised efficient inference execution mechanisms for them.

We hope that this effort has revealed some natural synergies between Generative AI systems and logic programming tools, ready to fill gaps like the lack of rigorous reasoning abilities of the LLMs, their lack of alignment to the user's intents and their known deficiencies on factuality.

References

[1] Joaquin 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), pp. 337–354, doi:10.1017/S1471068418000285.

[2] 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. Available at `https://www.sciencedirect.com/science/article/pii/S0004370221002009`.

[3] Harrison Chase (2022): *LangChain*. Available at `https://github.com/hwchase17/langchain`. Https://www.langchain.com/.

[4] Nuri Cingillioglu & Alessandra Russo (2020): *Learning Invariants through Soft Unification*, doi:10.48550/arXiv.1909.07328. arXiv:1909.07328.

[5] Luc De Raedt, Angelika Kimmig & Hannu Toivonen (2007): *ProbLog: A Probabilistic Prolog and Its Application in Link Discovery.* In: *IJCAI*, 7, pp. 2462–2467, doi:10.5555/1625275.1625673.

[6] William F. Dowling & Jean H. Gallier (1984): *Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae.* J. Log. Program. 1(3), pp. 267–284, doi:10.1016/0743-1066(84)90014-1. Available at `10.1016/0743-1066(84)90014-1`.

[7] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel et al. (2020): *Retrieval-augmented generation for knowledge-intensive nlp tasks*. Advances in Neural Information Processing Systems 33, pp. 9459–9474.

[8] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever & Karl Cobbe (2023): *Let's Verify Step by Step*, doi:10.48550/arXiv.2305.20050. arXiv:2305.20050.

[9] Zhan Ling, Yunhao Fang, Xuanlin Li, Zhiao Huang, Mingu Lee, Roland Memisevic & Hao Su (2023): *Deductive Verification of Chain-of-Thought Reasoning*, doi:10.48550/arXiv:2306.03872v3. arXiv:2306.03872.

[10] Jaron Maene & Luc De Raedt (2023): *Soft-Unification in Deep Probabilistic Logic*.

[11] Tuan Quoc Nguyen, Katsumi Inoue & Chiaki Sakama (2022): *Enhancing linear algebraic computation of logic programs using sparse representation*. New Generation Computing 40(1), pp. 225–254, doi:10.1007/s00354-021-00142-2.

[12] Nils Reimers & Iryna Gurevych (2019): *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. In Kentaro Inui, Jing Jiang, Vincent Ng & Xiaojun Wan, editors: EMNLP/IJCNLP (1), Association for Computational Linguistics, pp. 3980–3990, doi:10.18653/v1/D19-1410. Available at `http://dblp.uni-trier.de/db/conf/emnlp/emnlp2019-1.html#ReimersG19`.

[13] Chiaki Sakama, Katsumi Inoue & Taisuke Sato (2017): *Linear Algebraic Characterization of Logic Programs*. In Gang Li, Yong Ge, Zili Zhang, Zhi Jin & Michael Blumenstein, editors: Knowledge Science, Engineering and Management, Springer International Publishing, Cham, pp. 520–533, doi:10.1007/978-3-319-63558-3_44.

[14] Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart & Pascal Hitzler (2021): *Neuro-Symbolic Artificial Intelligence: Current Trends*, doi:10.48550/ARXIV.2105.05330. Available at `https://arxiv.org/abs/2105.05330`.

[15] Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie & Christopher D. Manning (2024): *RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval*, doi:10.48550/arXiv.2401.18059. arXiv:2401.18059.

[16] Thomas J. Schaefer (1978): *The complexity of satisfiability problems*. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC '78, Association for Computing Machinery, New York, NY, USA, p. 216?226, doi:10.1145/800133.804350. Available at `https://doi.org/10.1145/800133.804350`.

[17] Paul Tarau (2021): *Natlog: a Lightweight Logic Programming Language with a Neuro-symbolic Touch*. In Andrea Formisano, Yanhong Annie Liu, Bart Bogaerts, Alex Brik, Veronica Dahl, Carmine Dodaro, Paul Fodor, Gian Luca Pozzato, Joost Vennekens & Neng-Fa Zhou, editors: Proceedings 37th International Conference on Logic Programming (Technical Communications) , 20-27th September 2021, doi:10.4204/EPTCS.345.27.

[18] Paul Tarau (2023): *Full Automation of Goal-driven LLM Dialog Threads with And-Or Recursors and Refiner Oracles*:arXiv:2306.14077. doi:10.48550/arXiv.2306.14077. arXiv:2306.14077.

[19] Paul Tarau (2023): *Natlog: Embedding Logic Programming into the Python Deep-Learning Ecosystem*. In Enrico Pontelli, Stefania Costantini, Carmine Dodaro, Sarah Gaggl, Roberta Calegari, Artur D'Avila Garcez, Francesco Fabiano, Alessandra Mileo, Alessandra Russo & Francesca Toni, editors: Proceedings 39th International Conference on Logic Programming, Imperial College London, UK, 9th July 2023 - 15th July 2023, Electronic Proceedings in Theoretical Computer Science 385, Open Publishing Association, pp. 141–154, doi:10.4204/EPTCS.385.15.

[20] Paul Tarau (2024): *System Description: DeepLLM, Casting Dialog Threads into Logic Programs*. In Jeremy Gibbons & Dale Miller, editors: Functional and Logic Programming, Springer Nature Singapore, Singapore, pp. 117–134, doi:10.1007/978-981-97-2300-3_7.

[21] Paul Tarau & Eduardo Blanco (2021): *Interactive Text Graph Mining with a Prolog-Based Dialog Engine*. Theory Pract. Log. Program. 21(2), pp. 244–263, doi:10.1017/S1471068420000137.

[22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le & Denny Zhou (2022): *Chain of Thought Prompting Elicits Reasoning in Large Language Models*. CoRR abs/2201.11903, doi:10.48550/arXiv.2201.11903. arXiv:2201.11903.

Visual Graph Question Answering with ASP and LLMs for Language Parsing*

Jakob Johannes Bauer[1], Thomas Eiter[2], Nelson Higuera Ruiz[2], Johannes Oetsch[3]
[1] ETH Zürich, Rämistrasse 101, 8092 Zürich, Switzerland
[2] Vienna University of Technology (TU Wien), Favoritenstrasse 9–11, Vienna, 1040, Austria
[3] Jönköping University, Gjuterigatan 5, 551 11 Jönköping, Sweden
bjohannes@ethz.ch, {thomas.eiter,nelson.ruiz}@tuwien.ac.at, johannes.oetsch@ju.se

Visual Question Answering (VQA) is a challenging problem that requires to process multimodal input. Answer-Set Programming (ASP) has shown great potential in this regard to add interpretability and explainability to modular VQA architectures. In this work, we address the problem of how to integrate ASP with modules for vision and natural language processing to solve a new and demanding VQA variant that is concerned with images of graphs (not graphs in symbolic form). Images containing graph-based structures are an ubiquitous and popular form of visualisation. Here, we deal with the particular problem of graphs inspired by transit networks, and we introduce a novel dataset that amends an existing one by adding images of graphs that resemble metro lines. Our modular neuro-symbolic approach combines optical graph recognition for graph parsing, a pretrained optical character recognition neural network for parsing labels, Large Language Models (LLMs) for language processing, and ASP for reasoning. This method serves as a first baseline and achieves an overall average accuracy of 73% on the dataset. Our evaluation provides further evidence of the potential of modular neuro-symbolic systems, in particular with pretrained models that do not involve any further training and logic programming for reasoning, to solve complex VQA tasks.

1 Introduction

Visual Question Answering (VQA) [1] is concerned with inferring the correct answer to a natural language question in the presence of some visual input, such as an image or video, which typically involves processing multimodal input. VQA enables applications in, e.g., medicine, assistance for blind people, surveillance, and education [4].

Answer-Set Programming (ASP) [6] has shown great potential to add interpretability and explainability to modular VQA architectures in this context. As a knowledge representation and reasoning formalism with an intuitive modelling language, it can be used to describe how to infer answers from symbolic input provided by subordinate modules in a clear and transparent way [28, 5, 11, 10]. Another strength is that uncertainties from the underlying modules can be expressed using disjunctions (or choice rules), and we are not limited to inferring one answer, but several plausible ones in a nondeterministic manner [33]. Furthermore, using ASP in the VQA context is beneficial for explanation finding, as we have demonstrated in recent work [10].

In this work, we address the problem of how to integrate ASP with modules for vision and natural language processing to solve a new and demanding VQA variant that is concerned with images of graphs (not graphs in symbolic form). Visual representations of structures based on graphs are a popular and ubiquitous form of presenting information. It is almost surprising that VQA tasks where the visual input contains a graph have, to the best of our knowledge, not been considered so far.

*This work was partially funded from the Bosch Center for AI. Code and data can be found at https://github.com/pudumagico/NSGRAPH.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 15–28, doi:10.4204/EPTCS.416.2

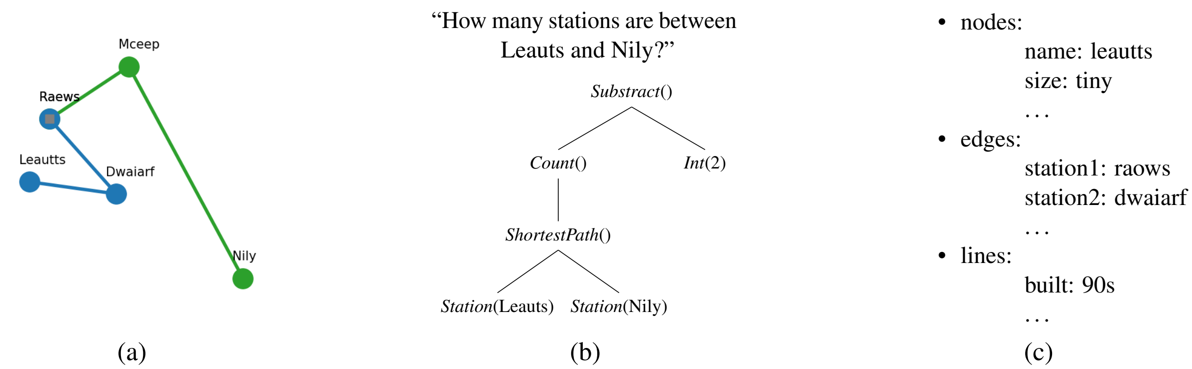


Figure 1: A CLEGR instance: metro graph with two lines, question with functional representation, and additional information. The task is to answer the question using the information provided.

We deal with the particular problem of graphs that resemble transit networks, and we introduce a respective dataset. It is based on the existing CLEGR dataset [22] that comes with a generator for synthetically producing vertex-labelled graphs that are inspired by metro networks. Additional structured information about stations and lines, e.g., how large a station is, whether it is accessible to disabled people, when the line was constructed, etc., is provided as background. The task is to answer natural language questions concerning such graphs. For example, a question may ask for the shortest path between two stations while avoiding those that have a particular property. An illustration of a graph and a question is shown in Fig. 1.

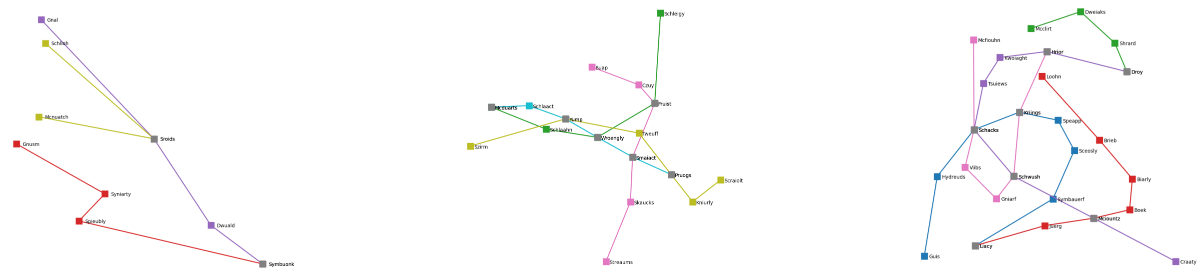
While purely symbolic methods suffice to solve the original CLEGR dataset with ease (we present one in this paper), we consider the more challenging problem of taking images of the graphs instead of their symbolic representations as input; an example is given in Fig. 1a. For the questions, we only consider those that can be answered with information that can be found in the image. The challenges to solve this VGQA dataset, which we call CLEGR^V, are threefold: (i) we have to parse the graph to identify nodes and edges, (ii) we have to read and understand the labels and associate them with nodes of the graph, and (iii) we have to understand the question and reason over the information extracted from the image to answer it accordingly.

Our solution takes the form of a modular (i.e., loosely coupled) neuro-symbolic model that combines the use of *optical graph recognition* (OGR) [2] for graph parsing, a pretrained *optical character recognition* (OCR) neural network [29] for parsing node labels, and, as mentioned above, ASP for reasoning. It operates in the following manner:

1. first, we use the OGR tool to parse the graph image into an abstract representation, structuring the information as sets of nodes and edges;
2. we use the OCR algorithm to obtain the text labels and associate them to the closest node;
3. then, we parse the natural language question;
4. finally, we use an encoding of the semantics of the question as a logic program which is, combined with the graph and the question in symbolic form, used to obtain the answer to the question with the help of an ASP solver.

This method serves as a first baseline and achieves an average accuracy of 73% on CLEGR^V.

We consider two methods to parse the natural language questions. The first one is to use regular expressions which are sufficient to parse the particular questions of the dataset. The second method uses Large Language Models (LLMs) based on the transformer architecture [31] to obtain a more robust



(a) (b) (c)

Figure 2: Examples of graphs of size small (2a), medium (2b), and large (2c).

solution that also generalises well to variants of questions that are not part of the dataset. Our approach to using LLMs follows related work [26] and relies on prompting an LLM to extract relevant ASP predicates from the question. We evaluated this approach on questions based on CLEGR and new questions obtained from a questionnaire.

The contribution of this paper is thus threefold:

(i) we demonstrate how ASP can be used as part of a modular VQA architecture able to tackle a challenging new problem concerned with images of graphs;

(ii) we introduce a new dataset to benchmark systems for VQA on images of graphs and evaluate our approach on it to create a first baseline; and

(iii) we evaluate various LLMs for question parsing to create a robust interface to the ASP encoding.

This work provides further evidence of the potential of modular neuro-symbolic systems, in particular with pretrained models and logic programming for reasoning, for solving complex VQA tasks. That our system does not require any training related to a particular set of examples—hence solving the dataset in a *zero-shot manner*—is a practical feature that hints to what may become customary as large pre-trained models are more than ever available for public use.

2 Visual Question Answering on Graphs

Graph Question Answering (GQA) is the task of answering a natural language question for a given graph in symbolic form. The graph consists of nodes and edges, but further attributes may be specified in addition. A specific GQA dataset is CLEGR [22], which is concerned with graph structures that resemble transit networks like metro lines. Its questions are ones that are typically asked about transit like "How many stops are between **X** and **Y**?". The dataset is synthetic and comes with a generator for producing instances of varying complexity.

Graphs come in the form of a YAML file containing records about attributes of the stations and lines. Each station has a name, a size, a type of architecture, a level of cleanliness, potentially disabled access, potentially rail access, and a type of music played. Stations can be described as relations over the aforementioned attributes. Edges connect stations but additionally have a colour, a line ID, and a line name. For lines, besides name and ID we have a construction year, a colour, and optional presence of air conditioning.

Example 1 *Examples of questions from the dataset are:*

- *Describe {Station} station's architectural style.*

- *How many stations are between {Station} and {Station}?*
- *Which {Architecture} station is adjacent to {Station}?*
- *How many stations playing {Music} does {Line} pass through?*
- *Which line has the most {Architecture} stations?*

For a full list of the questions, we refer the reader to the online repository of the dataset [22]. The answer to each question is of Boolean type, a number, a list, or a categorical answer. The questions in the dataset can be represented by functional programs, which allows us to decompose them into smaller and semantically less complex components. Figure 1 illustrates an example from the data set CLEGR that includes such a functional program: it consists of primitive operations organised as a tree that is recursively evaluated to obtain an answer.

Visual Graph Question Answering. Solving instances of the CLEGR dataset is not much of a challenge since all information is given in symbolic form, and we present a respective method later. But what if the graph is not available or given in symbolic form, but just as an image, as is commonly the case? We define *Visual Graph Question Answering* (VGQA) as a GQA task where the input is a natural language question on a graph depicted in an image.

The new VGQA dataset. We can in fact derive a challenging VGQA dataset from CLEGR by generating images of the transit graphs. To this end, we used the generator of the CLEGR dataset that can also produce images of the symbolic graphs. Each image shows stations, their names as labels in their proximity, and lines in different colours that connect them; an example is given in Fig. 1a. For the VGQA task, we drop all further symbolic information and consider only the subset of questions that can be answered with information from the graph image.

We call the resulting dataset CLEGR^V: it consists of graphs that fall into three categories: small (3 lines and at most 4 stations per line), medium (4 and at most 6 stations per line), and large (5 lines and at most 8 stations per line). We generate 100 graphs of each size accompanied by 10 questions per graph, with a median of 10 nodes and 8 edges for small graphs, 15 nodes and 15 edges for medium graphs, and 24 nodes with 26 edges for large ones. Figure 2 shows three graphs, one of each size. Although large metro networks will typically involve more stations than our graphs, those stations are typically arranged linearly on the lines which does not add to the complexity of the graph structure itself but can lead to cluttering.

3 Our Neuro-Symbolic Framework for VQA on Graphs

Our solution to the VGQA task, which we call NSGRAPH, is a modular neuro-symbolic system, whose modules are the typical ones for VQA, viz. a visual module, a language module, and a reasoning module, which we realise to fit the VQGA setting. Figure 3 illustrates the data flow of the inference process in NSGRAPH.

3.1 Visual Module

The visual model is used for graph parsing, which consists of two subtasks: (i) detection of nodes and edges, and (ii) detection of labels, i.e., station names.

We employ an optical graph recognition (OGR) system for the first subtask. In particular, we use a publicly available OGR script [9] that implements the approach due to Auer et al. [2]. The script takes

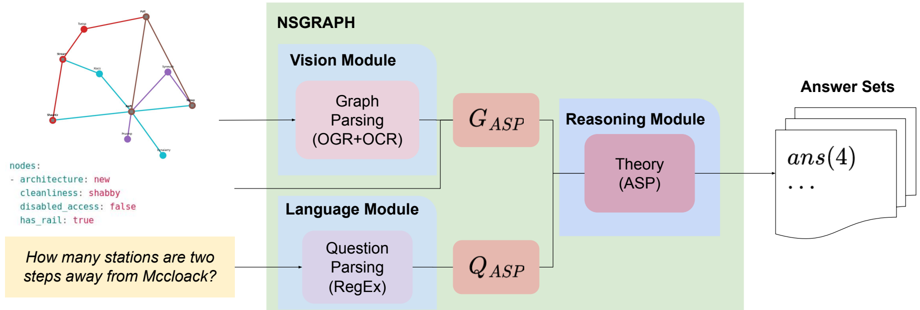


Figure 3: NSGRAPH system overview. The input is either an image of a graph or its symbolic description. The answer is generated by combining neural and symbolic methods.

an image as input and outputs the pixel coordinates of each detected node plus an adjacency matrix that contains the detected edges.

For the second subtask of detecting labels, we use an optical character recognition (OCR) system, namely, we use a pretrained neural network called EasyOCR [19] to obtain and structure the information contained in the graph image. The algorithm takes an image as input and produces the labels as strings together with their coordinates in pixels. We then connect the detected labels to the closest node found by the OGR system. Thereby, we obtain an abstract representation of the graph image as relations.

3.2 Language Module

The purpose of the language module is to parse the natural language question. It is written in Python and uses regular expressions to capture the variables in each type of question. There are in general 35 different question templates in CLEGR, some of which were shown in Example 1. They can be used to produce a question instances by replacing variables with names or attributes of stations, lines, or connections.

Example 2 *For illustration, the question template "How many stations are on the shortest path between S_1 and S_2?" may be instantiated by replacing S_1 and S_2 with station names that appear in the graph. We use regular expressions to capture those variables and translate the natural language question into a functional program, essentially a tree of operations, for that question. Continuing our example, we translate the template described above into the program*

```
end(3). countNodesBetween(2). shortestPath(1).
station(0,S1). station(0,S2).
```

where the the first numerical argument of each predicate imposes the order of execution of the associated operation and links the input of one operation to the output of the previous one. We can interpret this functional program as follows: the input to the shortest-path operation is two station names `S1` *and* `S2`*. Its outputs are the stations on the shortest path between* `S1` *and* `S2` *which are counted in the next step. The predicate* `end` *represents the end of the computation to yield this number as the answer to the question.*

All considered question types and their ASP question encodings are summarised in Table 1. Although this approach works well for all the questions in CLEGR, its ability to generalise to new types of questions is obviously limited; as a remedy, we discuss LLMs as an alternative to realise the language module in Section 4.

Table 1: ASP questions encodings for the twelve types of questions.

ASP Facts	Question
end(3). countNodesBetween(2). shortestPath(1). station(0,{}). station(0,{})	How many stations are between ([a-zA-Z]+) and ([a-zA-Z]+)?
end(2). withinHops(1, 2). station(0,{})	How many other stations are two stops or closer to ([a-zA-Z]+)?
end(2). paths(1). station(0,{}). station(0,{})	How many distinct routes are there between ([a-zA-Z]+) and ([a-zA-Z]+)?
end(2). cycle(1). station(0,{})	Is ([a-zA-Z]+) part of a cycle?
end(2). adjacent(1). station(0,{}). station(0,{})	Are ([a-zA-Z]+) and ([a-zA-Z]+) adjacent?
end(2). adjacentTo(1). station(0,{}).station(0,{})	Which station is adjacent to ([a-zA-Z]+) and ([a-zA-Z]+)?
end(2). commonStation(1). station(0,{}). station(0,{})	Are ([a-zA-Z]+) and ([a-zA-Z]+) connected by the same station?
end(2). exist(1). station(0,{})	Is there a station called ([a-zA-Z0-9]+)?
end(2). linesOnNames(1). station(0,{})	Which lines is ([a-zA-Z]+) on?
end(2). linesOnCount(1). station(0,{})	How many lines is ([a-zA-Z]+) on?
end(2). sameLine(1). station(0,{}). station(0,{})	Are ([a-zA-Z]+) and ([a-zA-Z]+) on the same line?
end(2). stations(1). line(0,{})	Which stations does ([a-zA-Z]+) pass through?

3.3 Reasoning Module

The third module consists of an ASP program that implements the semantics of the operations from the functional program of the question. Before we explain this reasoning component, we briefly review the basics of ASP.

Answer-Set Programming. ASP [6, 14] is a declarative logic-based approach to combinatorial search and optimisation with roots in knowledge representation and reasoning. It offers a simple modelling language and efficient solvers[1]. In ASP, the search space and properties of problem solutions are described by means of a logic program such that its models, called *answer sets*, encode the problem solutions.

An ASP program is a set of rules of the form $a_1 \mid \cdots \mid a_m \coloneq b_1, \ldots, b_n, \textit{not } c_1, \ldots, \textit{not } c_n,$ where all a_i, b_j, c_k are first-order literals and *not* is *default negation*. The set of atoms left of $\coloneq$ is the head of the rule, while the atoms to the right form the body. Intuitively, whenever all b_j are true and there is no evidence for any c_l, then at least some a_i must be true. The semantics of an ASP programs is given by its answer sets, which are consistent sets of variable-free (ground) literals that satisfy all rules and fulfil a minimality condition [15].

A rule with an empty body and a single head atom without variables is a *fact* and is always true. A rule with an empty head is a *constraint* and is used to exclude models that satisfy the body.

ASP provides further language constructs like choice rules, aggregates, and weak (also called soft) constraints, whose violation should only be avoided. For a comprehensive coverage of the ASP language and its semantics, we refer to the language standard [8].

[1] See, for example, `www.potassco.org` or `www.dlvsystem.com`.

Question Encoding. The symbolic representations obtained from the language and visual modules are first translated into ASP facts; we refer to them as G_{ASP} and Q_{ASP} in Fig.3, respectively. The functional program from a question (as introduced above) is already in a fact format. The graph is translated into binary atoms `edge/2` and unary atoms `station/1` as well. These facts combined with an ASP program that encodes the semantics of all CLEGR question templates can be used to compute the answer with an ASP solver.

Example 3 *Here is an excerpt of the ASP program that represents the functional program from above:*

```
end(3). countNodesBetween(2). shortestPath(1).
station(0,s). station(0,t).
```

These facts, together with ones for edges and nodes, serve as input to the ASP encoding for computing the answer as they only appear in rule bodies:

```
sp(T,S1,S2) :- shortestPath(T), station(T-1,S1),
                                station(T-1,S2), S1<S2'.
{ in_path(T,S1,S2) } :- edge(S1,S2), shortestPath(T).
reach(T,S1,S2) :- in_path(T,S1,S2).
reach(T,S1,S3) :- reach(T,S1,S2), reach(T,S2,S3).
:- sp(T,S1,S2), not reach(T,S1,S2).

cost(T,C) :- C = #count {S1,S2: in_path(T,S1,S2)}, shortestPath(T).
:~ cost(T,C). [C,T]

countedNodes(T,C-1) :- countNodesBetween(T),
                       shortestPath(T-1), cost(T-1,C).
ans(N) :- end(T), countedNodes(T,N).
```

The first rule expresses that if we see `shortestPath(T)` *in the input, then we have to compute the shortest path between station* `S1` *and* `S2`*. This path is produced by the next rule which non-deterministically decides for every edge if this edge is part of the path. The following two rules jointly define the transitive closure of this path relation, and the constraint afterwards enforces that station* `S1` *is reachable from* `S2` *on that path. We use a weak constraint to minimise the number of edges that are selected and thus enforce that we indeed get a shortest path. The number of edges is calculated using an aggregate expression to count. Finally, the penultimate rule calculates the number of stations on the shortest path, as it takes as input the nodes that came out of the shortest path from the previous step and counts them, and the last rule defines the answer to the question as that number. The complete encoding is part of the online repository of this project (*`https://github.com/pudumagico/NSGRAPH`*).*

3.4 Evaluation of NSGRAPH on CLEGR^V

NSGRAPH achieves 100% on the original GQA task, i.e., with graphs in symbolic form as input and with the complete set of questions. Here, the symbolic input is translated directly into ASP facts without the need to parse an image.

We summarise the results for the more challenging VGQA task on CLEGR^V in Table 2.[2] The task becomes more difficult with increasing size of the graphs, but still an overall accuracy of 73% is achieved. As we also consider settings where we replace the OCR, resp. the OGR module, with the ground truth

[2]We ran the experiments on a computer with 32GB RAM, 12th Gen Intel Core i7-12700K, and a NVIDIA GeForce RTX 3080 Ti, and we used clingo (v. 5.6.2) [13] as ASP solver.

Table 2: Accuracy of NSGRAPH on CLEGR^V for small, medium, and large sized graphs. For OCR+GT, we replaced the OGR input with its symbolic ground truth. Likewise, we use the ground truth for OCR for OGR+GT, and Full GT stands for ground truth only. We also report the total time for image parsing, resp. ASP reasoning, in seconds.

Graph Size	NSGRAPH	OCR+GT	OGR+GT	Full GT	parsing (s)	reasoning (s)
Small	80.9%	90.2%	83.1%	100%	923	2
Medium	71.0%	85.2%	72.7%	100%	1359	3
Large	67.2%	83.8%	70.5%	100%	2208	5
Overall	73.0%	86.4%	75.4%	100%	4490	10

as input, we are able to pinpoint the OGR as the main reason for wrong answers. The average run time to answer a question was $0.924\,s$ for small graphs, $1.36\,s$ for medium graphs, and $2.21\,s$ for large graphs. NSGRAPH is the first baseline for this VGQA dataset and further improvements a certainly possible, e.g., stronger OGR systems could be used.

4 Semantic Parsing with LLMs

LLMs like GPT-4 [24] are deep neural networks based on the transformer architecture [31] with billions of parameters that are trained on a vast amount of data to learn to predict the next token for a given text prompt. (A token is a sequences of textual characters like words or parts of words). Their capabilities for natural language processing are impressive. LLMs are typically instructed via text prompts to perform a certain task such as answering a question or translating a text, but they can also be used for semantic parsing a text into a formal representation suitable for further processing.

In this section, we outline and evaluate an approach to use LLMs to realise the language module of NSGRAPH in a more robust way than by using regular expressions. First, we outline the general method of prompting LLMs to extract ASP predicates from questions. Afterwards, we evaluated this method for different LLMs, including state-of-the-art API-based ones but also open-source models that are free and can be locally installed.

4.1 Prompt Engineering

A particularly useful feature of LLMs is that the user can instruct them for a task by providing a few examples as part of the input prompt without the need to retrain the model on task-specific data; a property of LLMs commonly referred to as *in-context learning*.

Our approach uses in-context learning to instruct the LLM to extract the ASP atoms needed to solve the reasoning task from a question. This idea is inspired by recent work on LLMs for language understanding [26]. To obtain an answer to a question Q, we

(i) create a prompt P(Q) that contains the question Q along with additional instructions and examples for ASP question encodings,
(ii) pass P(Q) as input to an LLM and extract the ASP question encoding from the answer, and
(iii) use extracted ASP facts together with the ASP rules described in the previous section to derive the answer.

The prompt P(Q) starts with a general pre-prompt that sets the stage for the task:

Table 3: Comparison of LLMs used in our evaluation.

Model	**Parameters**	**Open Source**	**Price**	**Company**	**Token Limit**
GPT-4	1.5×10^{12}	$\times$	USD 20 p/m	OpenAI	32 768
GPT-3.5	175×10^{9}	$\times$	free	OpenAI	4 096
Bard	1.6×10^{12}	$\times$	free	Google	2 048
GPT4ALL	7×10^{9}	✓	self hosted	Nomic AI	2 048
Vicuna 13b	13×10^{9}	✓	per request	Meta	2 048
Zephyr 7b	7×10^{9}	✓	free	HuggingFace H4	8 192

```
You are now a Question Parser that translates natural language
questions into ASP ground truths about different stations.
Output only the ground truths and nothing else. The stations to
be selected from are arbitrary.
```

Afterwards, we provide a number of examples that illustrate what is expected from the LLM. In particular, we used at least one not more than three examplesfor each type of question in the dataset to not exceed context limits. This amounts to 36 in-context examples in total.

Example 4 *For space reasons, we show here just the beginning of an example prompt:*

```
I now provide you with some examples on how to parse Questions:

Q: ‘‘How many stations are between Inzersdorf and Mainstation?’’
A: end(3).countNodesBetween(2).shortestPath(1).
station(0,‘‘Inzersdorf’’).station(0,‘‘Mainstation’’).

Q: ‘‘What is the amount of stations between Station A and
Station B?’’
A: end(3).countNodesBetween(2).shortestPath(1).
station(0,‘‘Station A’’).station(0,‘‘Station B’’).
...
```

Finally, the prompt contains the questions that should be answered:

```
Now provide the output for the following question:
What are the stations that lie on line 7?
```

4.2 Evaluation

We evaluated the method from the previous section to answer to following research questions:
(R1) Is the method suitable for realising the language component of NSGRAPH?
(R2) What is the trade-off between grand scale LLMs and smaller, more cost-efficient alternatives?
(R3) How well does the method generalise to questions formulated in a different way than in CLEGR?

Overview of used LLMs. We compared different models (GPT-4, GPT3.5, Bard, GPT4All, Vicuna 13b, and Zephyr 7b; cf. Table 3)[3] on the semantic parsing task.

[3] https://openai.com/research/gpt-4; https://platform.openai.com/docs/models/gpt-3-5; https://bard.google.com/; https://gpt4all.io/index.html;https://huggingface.co/lmsys/vicuna-13b-v1.3; https://huggingface.co/HuggingFaceH4/zephyr-7b-beta.

Table 4: Results of the evaluation on the CLEGR$^+$ dataset.

Model	full match	contains solution	task missed	no answer
GPT-4	85%	0%	15%	0%
GPT-3.5	42%	8%	50%	0%
Bard	0%	76%	24%	0%
GPT4ALL	0%	23%	77%	0%
Vicuna 13b	8%	24%	34%	34%
Zephyr 7b	0%	61%	21%	18%

GPT-4 is the latest model developed by OpenAI with 1.5 trillion parameters and a context limit of 32 768 tokens. For a price of USD 20 per month, the ChatGPT Plus offer can be subscribed, allowing users to send up to 50 requests in a three-hour time frame to a hosted version of GPT-4.

GPT3.5 is the predecessor of OpenAIs GPT-4 and is available online for free. It uses 175 billion parameters and is capable of contexts of 4 096 tokens.

Bard is Google's counterpart to OpenAI's dominant LLMs, using slightly more parameters than GPT-4 but has a context window of only 2 048 tokens. It is free to its users; however, all EU states are currently excluded from using the service due to copyright concerns.

GPT4All is an open source model that only needs 7 billion parameters. With a context limit of 2 048 tokens, it competes with Google Bard; however, there is no official hosted service to run this LLM. It was developed using the open source weights of Alpaca, a model developed and released by Meta. GPT-4 served as a training data generator for this model, making it a cheap alternative to expensive large-scale models.

Vicuna 13b was developed and open-sourced by Meta and comes with 13 billion parameters and a context window of 2 018 tokens. It serves as a middle ground between large-scale LLMs and small alternatives such as GPT4All. It is not hosted on an official server, but there are external services that host this model and even offer fine-tuning to user specific use cases.

Zephyr 7b (β) is a fine-tuned version of the Mistral 7B model. It was developed by the Hugging Face H4 team and is published under the MIT license.

Datasets. We created two datasets for our evaluation: CLEGR$^+$ and CLEGR-Human. The former is a straight-forward hand-crafted extension of the questions from the original CLEGR dataset. Besides original questions that can be parsed with regular expressions, the dataset also contains versions where words are replaced with synonyms and the position of words is slightly changed, as well as questions that entirely rephrase the original ones. For example, "Are stations A and B on the same line?" could be rephrased as "Can I reach station A from station B without line change?". The CLEGR$^+$ dataset consists of 74 questions in total.

CLEGR-Human is a dataset that was created using an online survey. The survey takers were presented with a metro map and a couple of example questions. After that, they had the task of formulating further questions such as "Ask about the distance between Station A and Station B" and answering their own questions. This enables cross-peer validation by having other users evaluate the same question and compare their answers. Each surveyor had to answer a total of 12 questions; 27 people from Austria, Switzerland, and Germany aged between 18 and 33 years completed the survey, 22 of which were students. The dataset consists of 324 questions in total.

Table 5: Results of the evaluation on the CLEGR-Human dataset.

Model	full match	contains solution	task missed	no answer
GPT-4	94%	0%	6%	0%
GPT-3.5	38%	7%	55%	0%
Bard	0%	78%	22%	0%
GPT4ALL	0%	16%	84%	0%
Vicuna 13b	4%	19%	46%	31%
Zephyr 7b	0%	72%	17%	11%

Results and Discussion. The results of our evaluation are summarised in Table 4 for CLEGR$^+$ and in Table 5 for CLEGR-Human. We classified the answers produced by the LLMs into four categories:

- *full match*: the response matches exactly the set of expected atoms;
- *contains solution*: the expected atoms can be extracted from the respone;
- *task missed*: the response contains some text but not the expected atoms;
- *no answer*: the response consists of only whitespace characters.

Note that "full match" as well as "contains solution" can be used for the ASP reasoning task, while answers from the other categories cannot be used.

GPT-4 performed best among the considered LLMs as it produced 85% completely correct responses on CLEGR$^+$ and even 94% on CLEGR-Human. It also always provided an answer to the prompt. GPT-3.5 invented new predicates for half of the questions. For the remaining ones, its response matched exactly or contained the solution. Vicuna often did not give a proper response due to context overflow and trails behind GPT-4 and GPT-3.5 also in terms of correct answers. Google Bard never got the exact solution due to extensive additional explanations for all predicates no matter the prompt. However, the responses contained the solution in about three quarters of the cases. In this regard, it is only outmatched by GPT-4. This performance is similar to that of the much smaller open-source model Zephyr 7b, which is trailing only slightly behind. The responses of GPT4All contain the correct solution for only 23% (CLEGR$^+$) and 16% (CLEGR-Human) of the questions.

We answer our initial research questions therefore as follows: At least GPT-4 is suitable for realising the language component with an acceptable trade-off between accuracy and ability to generalise (R1). Although GPT-4 exhibits the best overall performance, especially the free and much smaller Zephyr model shows promising results (R2). Throughout, the LLMs perform similarly on CLEGR$^+$ and CLEGR-Human, which showcases the strength of LLMs for language processing without the need for context-specific training (R3).

5 Related Work

Our approach builds on previous work [11], where we introduced a neuro-symbolic method for VQA in the context of the CLEVR dataset [20] using a reasoning component based on ASP inspired by NSVQA [34]. The latter used a combination of RCNN [27] for object detection, an LSTM [16] for natural language parsing, and Python as a symbolic executor to infer the answer. The vision and language modules in these previous approaches were trained for the datasets. As compared to these datasets the number of questions obtained from the questionnaires to build our dataset is small, it would be hardly possible to effectively train an LSTM on them. It is a particular strength of our work that we resort to LLMs that do not require any further training.

We also mention the neural and end-to-end trainable MAC system [17] that achieves very promising results in VQA datasets, provided there is enough data available to train the system. A recent approach that combines large pretrained models for images and text in combination with symbolic execution in Python is ViperGPT [30]; complicated graph images are not handled well by pretrained vision-language models, however.

A characteristic of NSGRAPH is that we use ASP for reasoning, an idea that was also explored in previous work [28, 5, 11, 10]. Outside of the context of VQA, ASP has been applied for various neuro-symbolic tasks such as segmentation of laryngeal images [7], and discovery of rules that explain sequences of sensory input [12]. Barbara et al. [3] describe a neuro-symbolic approach that involves ASP for visual validation of electric panels where a component graph from an image is matched against its specification. This is an example of another interesting application that involves images of graphs and our approach could be used to contribute question-answering capabilities in such a setting.

In passing, it should be noted that there are also systems that can be used for neuro-symbolic learning, e.g., by employing semantic loss [32], which means that they use the information produced by the reasoning module to improve the learning tasks of the neural networks involved [33, 23].

Our approach to using LLMs to extract predicates for the downstream reasoning task is inspired by recent work by Rajasekharan et al. [26]. They proposed the STAR framework, which consists of LLMs and prompts for extracting logical predicates in combination with an ASP knowledge base. The authors applied STAR to different problems requiring qualitative reasoning, mathematical reasoning, as well as goal-directed conversation. Going one step further, Ishay et al. [18] introduced a method to translate problems formulated in natural language into complete ASP programs. This method requires multiple prompts, each responsible for a subtask such as identifying constant symbols, forming predicates, and transforming the specification into rules. The idea to apply LLMs to parse natural language into a formal language suitable for automated reasoning is also found outside the context of ASP, e.g., work by Liu et al. [21], who use prompting techniques to translate text into the Planning Domain Definition Language.

6 Conclusion

We addressed the relevant the problem of integrating ASP with vision and language modules to solve a new VQA variant that is concerned with images of graphs. For this task, we introduced a respective dataset that is based on an existing one for graph question answering on transit networks, and we presented NSGRAPH, a modular neuro-symbolic model for VGQA that combines neural components for graph and question parsing and symbolic reasoning with ASP for question answering. We studied LLMs for the question parsing component to improve how well our method generalises to unseen questions. NSGRAPH has been evaluated on the VGQA dataset and therefore constitutes a first baseline for the novel dataset.

The advantages of a modular architecture in combination with logic programming are that the solution is transparent, interpretable, explainable, easier to debug, and components can be replaced with better ones over time in contrast to more monolithic end-to-end trained models. Our system notably relies on pretrained components and thus requires no additional training. With the advent of large pretrained models for language and images such as GPT-4 [24] or CLIP [25], such architectures, where symbolic systems are used to control and connect neural ones, may be seen more frequently.

For future work, we plan to look into better alternatives for the visual module that is more suitable for complicated images of graphs, which is currently the limiting factor. Another future direction is to work with real-world metro networks for which currently no VQA datasets exist.

References

[1] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick & Devi Parikh (2015): *VQA: Visual Question Answering*. In: *2015 IEEE International Conference on Computer Vision*, IEEE Computer Society, pp. 2425–2433, doi:10.1109/ICCV.2015.279.

[2] Christopher Auer, Christian Bachmaier, Franz J. Brandenburg, Andreas Gleißner & Josef Reislhuber (2013): *Optical Graph Recognition*. In: *Graph Drawing*, Springer Berlin Heidelberg, pp. 529–540, doi:10.1007/978-3-642-36763-2_47.

[3] Vito Barbara, Massimo Guarascio, Nicola Leone, Giuseppe Manco, Alessandro Quarta, Francesco Ricca & Ettore Ritacco (2023): *Neuro-Symbolic AI for Compliance Checking of Electrical Control Panels*. *Theory Pract. Log. Program.* 23(4), pp. 748–764, doi:10.1017/S1471068423000170.

[4] Silvio Barra, Carmen Bisogni, Maria De Marsico & Stefano Ricciardi (2021): *Visual question answering: Which investigated applications?* *Pattern Recognit. Lett.* 151, pp. 325–331, doi:10.1016/j.patrec.2021.09.008.

[5] Kinjal Basu, Farhad Shakerin & Gopal Gupta (2020): *AQuA: ASP-Based Visual Question Answering*. In: *22nd International Symposium on Practical Aspects of Declarative Languages (PADL 2020)*, *LNCS* 12007, Springer, pp. 57–72, doi:10.1007/978-3-030-39197-3_4.

[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] Pierangela Bruno, Francesco Calimeri, Cinzia Marte & Marco Manna (2021): *Combining Deep Learning and ASP-Based Models for the Semantic Segmentation of Medical Images*. In: *5th International Joint Conference on Rules and Reasoning*, *LNCS* 12851, Springer, pp. 95–110, doi:10.1007/978-3-030-91167-6_7.

[8] 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.

[9] Filip Chodziutko & Kacper Nowakowski (2020): *Optical Graph Recognition (OGR) - script*. Available at `https://github.com/praktyka-zawodowa-2020/optical_graph_recognition`.

[10] Thomas Eiter, Tobias Geibinger, Nelson Higuera & Johannes Oetsch (2023): *A Logic-based Approach to Contrastive Explainability for Neurosymbolic Visual Question Answering*. In: *32nd International Joint Conference on Artificial Intelligence (IJCAI 2023)*, ijcai.org, pp. 3668–3676, doi:10.24963/IJCAI.2023/408.

[11] Thomas Eiter, Nelson Higuera, Johannes Oetsch & Michael Pritz (2022): *A Neuro-Symbolic ASP Pipeline for Visual Question Answering*. *TPLP* 22(5), pp. 739–754, doi:10.1017/S1471068422000229.

[12] Richard Evans, José Hernández-Orallo, Johannes Welbl, Pushmeet Kohli & Marek J. Sergot (2021): *Making sense of sensory input*. *Artif. Intell.* 293, p. 103438, doi:10.1016/j.artint.2020.103438.

[13] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub & Philipp Wanko (2016): *Theory Solving Made Easy with Clingo 5*. In: *ICLP 2016*, *OASIcs* 52, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, pp. 2:1–2:15, doi:10.4230/OASIcs.ICLP.2016.2.

[14] 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.

[15] 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.

[16] Sepp Hochreiter & Jürgen Schmidhuber (1997): *Long Short-Term Memory*. *Neural Comput.* 9(8), pp. 1735–1780, doi:10.1162/neco.1997.9.8.1735.

[17] Drew A. Hudson & Christopher D. Manning (2018): *Compositional Attention Networks for Machine Reasoning*. In: *6th International Conference on Learning Representations*, OpenReview.net. Available at `https://openreview.net/forum?id=S1Euwz-Rb`.

[18] Adam Ishay, Zhun Yang & Joohyung Lee (2023): *Leveraging Large Language Models to Generate Answer Set Programs*. In: *Proceedings of KR 2023*, pp. 374–383, doi:10.24963/kr.2023/37.

[19] Jaided AI (2022): *EasyOCR*. Available at `https://https://github.com/JaidedAI/EasyOCR`.

[20] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick & Ross B. Girshick (2017): *CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning*. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, pp. 1988–1997, doi:10.1109/CVPR.2017.215.

[21] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas & Peter Stone (2023): *LLM+P: Empowering Large Language Models with Optimal Planning Proficiency*. *CoRR* abs/2304.11477, doi:10.48550/arXiv.2304.11477. arXiv:2304.11477.

[22] D. Mack & A. Jefferson (2018): *CLEVR graph: A dataset for graph question answering*. Available at `https://github.com/Octavian-ai/clevr-graph`.

[23] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester & Luc De Raedt (2021): *Neural probabilistic logic programming in DeepProbLog*. *Artif. Intell.* 298, p. 103504, doi:10.1016/J.ARTINT.2021.103504.

[24] OpenAI (2023): *GPT-4 Technical Report, arXiv:2303.08774 [cs.CL]*.

[25] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger & Ilya Sutskever (2021): *Learning Transferable Visual Models From Natural Language Supervision*. In: *38th International Conference on Machine Learning*, *Proceedings of Machine Learning Research* 139, PMLR, pp. 8748–8763.

[26] Abhiramon Rajasekharan, Yankai Zeng, Parth Padalkar & Gopal Gupta (2023): *Reliable Natural Language Understanding with Large Language Models and Answer Set Programming*. *CoRR* abs/2302.03780, doi:10.48550/arXiv.2302.03780. arXiv:2302.03780.

[27] Shaoqing Ren, Kaiming He, Ross B. Girshick & Jian Sun (2017): *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. *IEEE Trans. Pattern Anal. Mach. Intell.* 39(6), pp. 1137–1149, doi:10.1109/TPAMI.2016.2577031.

[28] Heather Riley & Mohan Sridharan (2019): *Integrating Non-monotonic Logical Reasoning and Inductive Learning With Deep Learning for Explainable Visual Question Answering*. *Frontiers Robotics AI* 6, p. 125, doi:10.3389/FROBT.2019.00125.

[29] Abin M Sabu & Anto Sahaya Das (2018): *A Survey on various Optical Character Recognition Techniques*. In: *2018 Conference on Emerging Devices and Smart Systems (ICEDSS)*, pp. 152–155, doi:10.1109/ICEDSS.2018.8544323.

[30] Dídac Surís, Sachit Menon & Carl Vondrick (2023): *ViperGPT: Visual Inference via Python Execution for Reasoning*. *CoRR* abs/2303.08128, doi:10.48550/arXiv.2303.08128. arXiv:2303.08128.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser & Illia Polosukhin (2017): *Attention is All you Need*. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pp. 5998–6008.

[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: *35th International Conference on Machine Learning*, *Proceedings of Machine Learning Research* 80, PMLR, pp. 5498–5507.

[33] Zhun Yang, Adam Ishay & Joohyung Lee (2020): *NeurASP: Embracing Neural Networks into Answer Set Programming*. In: *29th International Joint Conference on Artificial Intelligence, IJCAI 2020*, ijcai.org, pp. 1755–1762, doi:10.24963/ijcai.2020/243.

[34] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli & Josh Tenenbaum (2018): *Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding*. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018*, pp. 1039–1050.

LLM+Reasoning+Planning for Supporting Incomplete User Queries in Presence of APIs

Sudhir Agarwal Anu Sreepathy
Intuit AI Research, Mountain View, CA, USA
{sudhir_agarwal, anu_sreepathy}@intuit.com

David H. Alonso Prarit Lamba
Intuit Inc., Mountain View, CA, USA
{david_haroalonso, prarit_lamba}@intuit.com

Recent availability of Large Language Models (LLMs) has led to the development of numerous LLM-based approaches aimed at providing natural language interfaces for various end-user tasks. These end-user tasks in turn can typically be accomplished by orchestrating a given set of APIs. In practice, natural language task requests (user queries) are often incomplete, i.e., they may not contain all the information required by the APIs. While LLMs excel at natural language processing (NLP) tasks, they frequently hallucinate on missing information or struggle with orchestrating the APIs. The key idea behind our proposed approach is to leverage logical reasoning and classical AI planning along with an LLM for accurately answering user queries including identification and gathering of any missing information in these queries. Our approach uses an LLM and ASP (Answer Set Programming) solver to translate a user query to a representation in Planning Domain Definition Language (PDDL) via an intermediate representation in ASP. We introduce a special API "get_-info_api" for gathering missing information. We model all the APIs as PDDL actions in a way that supports dataflow between the APIs. Our approach then uses a classical AI planner to generate an orchestration of API calls (including calls to get_info_api) to answer the user query. Our evaluation results show that our approach significantly outperforms a pure LLM based approach by achieving over 95% success rate in most cases on a dataset containing complete and incomplete single goal and multi-goal queries where the multi-goal queries may or may not require dataflow among the APIs.

1 Introduction

Customers of large organizations have a variety of questions or requests (collectively known as queries in the following) pertaining to the organization's domain of operation. Providing relevant and accurate responses to such user queries is critical and requires a thorough analysis of the user's context, product features, domain knowledge, and organization policies. The user queries may encompass a variety of types - data lookup and aggregation queries, help requests, how-to questions, record update requests or a combination of these types.

Recently, transformer-based large language models (LLMs) have shown wide success on many natural language understanding and translation tasks, also demonstrating some general database querying [5, 16, 25] and reasoning and planning [14, 13, 18, 30] capability on diverse tasks without having to be retrained. However, the data and knowledge required for accurately answering customers' queries are partly or completely organization internal and not available to LLMs trained on publicly available data. Even in case of organization internal LLM deployments, it is often not feasible to give LLMs direct access to databases for various security and privacy reasons. In lieu of that, organizations develop APIs to make these internal artifacts programmatically accessible to the organization's applications.

Several frameworks and techniques have been proposed for answering user queries using a combination of LLM and tools/APIs, e.g. LangChain [3], Gorilla [21], ToolFormer [23], and TravelPlanner [19, 27]. However, such frameworks rely on LLMs for selecting and composing tools and as a result

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 29–58, doi:10.4204/EPTCS.416.3

either do not scale well beyond a small set of APIs/tools or have limited planning and API orchestration capability. These weaknesses limit the use of such frameworks for practical industrial applications.

To address these limitations, some recent works have investigated the use of an external classical planner along with an LLM. Given a description of the possible initial states of the world, a description of the desired goals, and a description of a set of possible actions, the classical planning problem involves synthesizing a plan that, when applied to any initial state, generates a state which contains the desired goals (goal state) [9]. The approaches presented in [1, 18] have demonstrated that utilizing an LLM to create the task PDDL (a representation of a user query as a planning problem in Planning Domain Definition Language) from a natural language planning task description, and then utilizing an external classical planner to compute a plan, yields better performance than relying solely on an LLM for end-to-end planning. However, these approaches have been shown to support only classical planning tasks, which hinders their use for answering user queries in the presence of APIs.

Furthermore, all the above mentioned approaches assume complete user queries, i.e., queries that contain all the required information for computing an answer to the query. In practice however, user queries are often incomplete. In general, detecting and gathering missing information depends on the granularity of the underlying atomic actions or APIs as well as dataflow among them at runtime. For example, if a user wants to book a flight and provides the source and destination airports information but the flight booking API requires the travel date as well, the user query is considered incomplete with respect to the available APIs. The AutoConcierge framework [31] can detect missing information for a pre-defined goal assuming that the required information for accomplishing the goal is known a-priori. However, there is still a need for an approach that can handle different kinds of possibly incomplete queries.

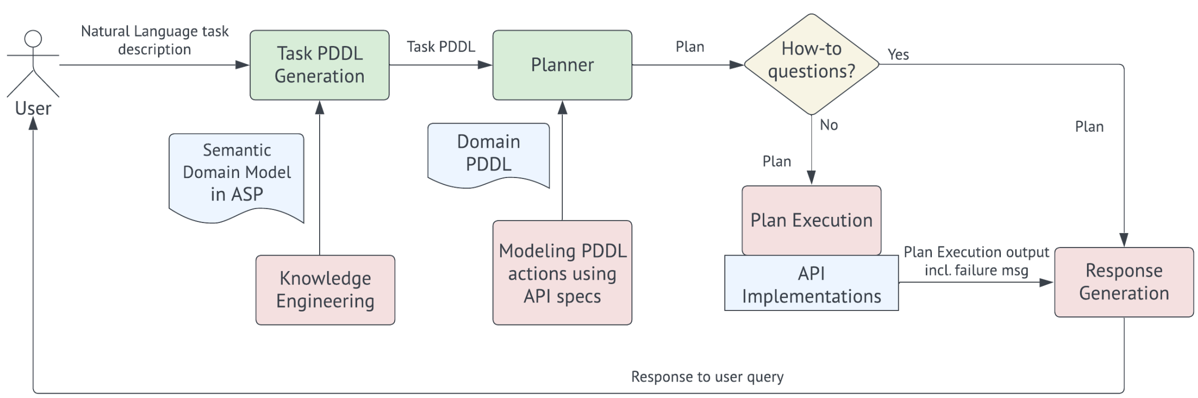


Figure 1: Overview of user query answering using LLMs and Classical Planning

Figure 1 presents the high level architecture of our approach for supporting several kinds of user queries using a given set of APIs. We translate a user query to a task PDDL (query's representation in PDDL) and use a classical AI planner for orchestrating APIs (plan) for the generated task PDDL. The plan execution component executes the plan by invoking the APIs in the specified order. For how-to questions, the plan is not executed but sent to the response generation component. Finally, the response generation component generates the overall response to be sent to the user from the individual outputs of the API calls. In this paper, our focus is on the Task PDDL Generation and Planner components in particular for supporting incomplete user queries.

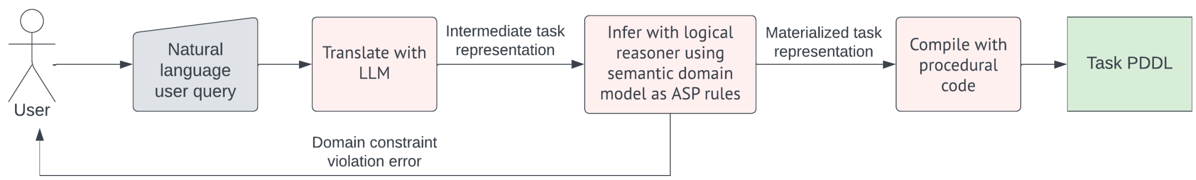


Figure 2: Steps for translating a user query to task PDDL

Figure 2 illustrates our process of translating a user query to task PDDL by using a novel combination of an LLM and logical reasoning using Answer Set Programming (ASP) [2, 17, 7]. We use an LLM to generate an intermediate representation of a user query in ASP. Our LLM prompting technique is generic and allows a set of possible user goal specifications to be plugged in. This step is described in Section 3.1. Such intermediate representations allow us to use an ASP solver to deterministically infer additional information, detect inconsistencies in user queries with respect to domain constraints, and bridge the syntactic and semantic heterogeneities between a user query and the target task PDDL. We refer to the union of facts in the intermediate representation and the inferred information as materialized representation of the user query. This step is described in Section 3.2. In cases, where an intermediate representation violates any domain constraints, the materialized representation contains corresponding errors. In these cases, we send the errors back to user. In other cases, we obtain the task PDDL by converting the materialized representation which is in the ASP syntax to PDDL syntax using deterministic procedural code. This step is described in Section 4.1.

In the next step, we use a classical planner with the task PDDL and an offline created PDDL domain model which includes domain concepts as predicates and specification of the APIs as PDDL actions in terms of these domain predicates (Section 2). In addition to the given set of functionality providing APIs, we introduce a special API `get_info_api` for gathering missing information from the user or an external system at runtime in order to support incomplete queries. The planner returns a plan (including calls to get_info_api in case of incomplete queries) such that the execution of the plan computes the answer to the user query. The plan generation step is described in Section 4.2.

Since there aren't any benchmark datasets of incomplete queries to be answered using APIs, we generated a dataset containing single goal and multi-goal complete and incomplete natural language queries based off a set of APIs described in Section 2. We refer to a domain concept in a user query as a goal. Our evaluation results on this dataset show that our approach significantly outperforms a pure LLM based approach by achieving over 95% success rate in most cases.

2 Specification of APIs as PDDL Actions

Throughout this paper, we use the following APIs which are derived from the set of publicly available Intuit Developer APIs[1] for experimental purposes. • *Profit and loss report API*: Generates profit and loss report for a given time period. • *Expense and spend report API*: Generates expense and spend report for a given time period. • *Invoices and sales report API*: Generates invoices and sales report for a given time period. • *Charge lookup API*: Generates detailed report for a given charge amount on a given date. • *Help API*: Provides answer to a given how-to question in a product. • *Contact API*: Connects customer to a

[1] `https://developer.intuit.com/app/developer/homepage`

human customer agent over a given communication channel for a conversation on a given topic • *Advice API*: Provides advice for a given personal finance or a small business relation question. • *Create invoice API*: Creates a new invoice for given amount and invoice detail. • *Update customer API*: Updates a customer profile with new first name, last name, phone, and email.

In order to be able to use a classical planner for computing an orchestration of available APIs, we model each available API as an action in PDDL. PDDL serves as a standardized encoding of classical planning problems [8, 11]. A PDDL representation of an action consists of the action's pre-conditions and effects defined using logical formulas with domain predicates, local variables (action's parameters) and constants. Note that unlike familiar procedural programming languages, PDDL actions' outputs are also declared as part of action's parameters. The PDDL representation of a planning problem is typically separated into two files: a domain PDDL file and a task PDDL file, both of which become inputs to the planner. Broadly, the domain PDDL file includes declaration of object types, predicates, and specification of actions. The task PDDL file provides a list of objects to ground the domain, and the problem's initial state and goal conditions defined in terms of the predicates.

Below the PDDL representation of the profit&loss API as action `profit_loss_api`. The action generates a profit and loss report for given time period. The pre-condition of the action means that variables `?in1` and `?in2` have type `date` as well as have a value (i.e., they are not NULL). The `?out` var represents the generated report. The pre-condition also includes that the `?out` must have the type `profit_loss_report` but must not have a value (indicating that `?out` doesn't represent an already previously generated report). The effects of the action mean that after execution of the action the value of `?out` is set. Furthermore, the effects mean that after the execution of the action, the generated report `?out` has `?in1` and `?in2` as start date and end date of the generated report `?out` respectively.

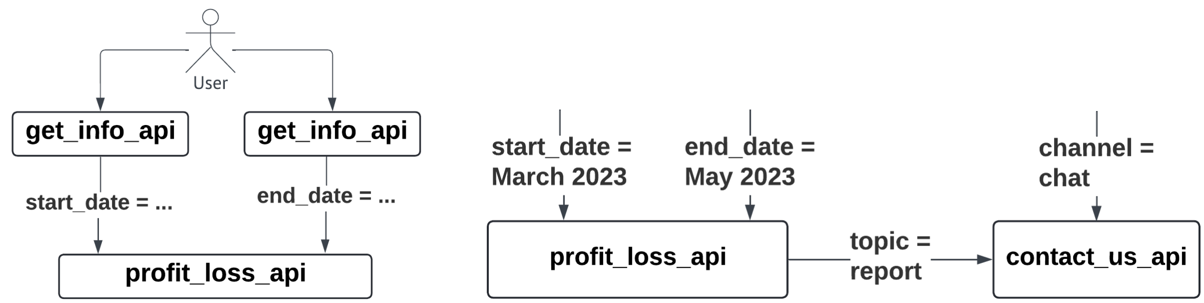
```
(:action profit_loss_api
    :parameters (?in1 - var ?in2 - var ?out - var)
    :precondition (and (has_type ?in1 date) (has_value ?in1)
       (has_type ?in2 date) (has_value ?in2)
       (has_type ?out profit_loss_report) (not (has_value ?out)))
    :effect (and (start_date ?out ?in1)
       (end_date ?out ?in2) (has_value ?out)))
```

A classical planner will find the above action for a user goal requesting a profit and loss report for given start and end dates. However, if the start date or the end date or both are not provided, a planner will fail to find `profit_loss_api` as relevant action.

We address this problem by introducing a special action `get_info_api` to gather information from the user or an external system at runtime. We model `get_info_api` as a PDDL action as shown below. The `get_info_api` action requires a variable of a type that is not set and ensures that it is set after the execution of `get_info_api`.

```
(:action get_info_api
    :parameters (?in_var - var ?in_type - var_type)
    :precondition (and (has_type ?in_var ?in_type)
        (not (has_value ?in_var)))
    :effect (and (has_value ?in_var)))
```

This modeling of `get_info_api` enables a planner to include get_info_api calls in the plan for gathering missing information. For example, for the query in Figure 3a we aim at detecting the profit & loss report API, and asking the user for the missing report time period. Similarly, in case of a more



(a) Plan for query *Show me my profit and loss report*

(b) Plan for query *Can I see my profit and loss statement from March to May 2023? I would like to discuss my profits further over chat*

Figure 3: Plan for an incomplete and a complete query

complex user query in Figure 3b, we aim at detecting the profit & loss report API and the contact API as well as the profit & loss report as the conversation topic with the customer agent.

For the purpose of this paper, we have modeled the domain PDDL manually. Efficient authoring of domain PDDL is out of scope of this work. However, we would like to point that approaches such as [10] may be leveraged for (semi-) automatically generating the domain PDDL for large domains. Refer to Appendix B.1 for the specification of all APIs in our dataset.

3 User Query to ASP Representation

As illustrated in Figure 2, in order to generate task PDDL for a user query, in the first step, we use an LLM for translating the user query to an intermediate representation in ASP. The main reason behind this step is that LLMs perform well on such translation tasks while they hallucinate when they are also required to generate logically derivable information [29, 15, 4, 26]. In the second step, we use a logical reasoner for inferring other information similar to approaches presented in [22, 28, 1].

3.1 User Query to Intermediate Representation

We construct the LLM prompt with the following steps for translating user query to an intermediate representation in ASP.

Step 1: Define a set of supported goals. The set of goals doesn't need to have 1:1 correspondence with the set of APIs. But, the set of goals corresponds to expected user requests. Such a modeling enables decoupling of user requests from APIs as the end users can not be expected to be familiar with the APIs (cf. OpenAI function calling approach [2]).

Step 2: Describe argument types. For each argument of the supported goals, define the type by giving a few examples or the set of possible values as appropriate. Below example defines argument types for date period and communication channel. See Appendix A.1 for definition of all argument types for our dataset.

[2] `https://platform.openai.com/docs/guides/function-calling`

```
arg_type_date_period = {"examples": {"nov 2023": ("11/01/2023",
"11/30/2023"), "fy21": ("01/01/2021", "12/31/2021"), ...}

arg_type_comm_channel = {"possible_values": ["video", "chat", "phone"]}
```

Step 3: Describe domain goals. Describe each goal using a name, description and required information for the goal. Refer to Appendix A.2 for complete list of supported domain goals.

```
{"name": "goal_1", "description": "request for report on profit, loss,
earnings, business insights, revenue, figures.", "required information":
[{"name": "report_period", "description": "time period of the requested
report defined by start and end dates.", "type" : arg_type_date_period}]}
```

Step 4: Define instructions. We instruct the LLM to extract goals and required information from the user query.

```
Given goal types with their required information. Extract from the
provided user query:
1. The one or more goals of the query from the given set of goals.
Represent each extracted goal <x> of type <T> as "_goal(<x>, <T>).".
2. If the user query contains any required information for the extracted
goal, then extract that too. While doing so, if possible values
are defined for the argument, then choose one from them if applicable.
```

Step 5: Construct LLM prompt. LLM prompt also includes a few in-context examples that are independent of the domain of our dataset. Refer to Appendix A.3 for complete list of in-context examples.

```
<Instructions as described above>

Below a few examples of goals, text and the answer.

<As in Appendix A.3>

Goals: """ <Domain goals as described above.> """

Text: """ <user query> """

Answer:
```

Below are a few example queries and their respective intermediate representations in ASP as returned by the LLM.

Example 1. *Show me 2023 Q1 detailed expense report.*
```
_goal(x, goal_2).
_report_period(x, ("01/01/2023",
    "03/31/2023")).
```

Example 2. *Provide me with the profit and loss statement for the previous quarter and put me on a phone call with a representative to discuss it.*
```
_goal(x, goal_1).
_report_period(x, ("07/01/2024",
    "09/30/2024")).
_goal(y, goal_4).
_contact_topic(y, x).
_contact_channel(y, "phone").
```

Example 3. *Profit and loss report.*
```
_goal(x, goal_1).
```

Example 4. *I want to chat with a representative.*
```
_goal(x, goal_6).
_contact_channel(x, "chat").
```

Example 5. *Show me expense report from July 2024 to Jan 2024.*
```
_goal(x, goal_2).
_report_period(x, ("07/01/2024",
    "01/31/2024")).
```

The query in Example 1 is a complete query. The query in Example 2 is a complete query with two goals and dataflow. The profit & loss report `x` is the topic of the conversation for the contact `y`. The queries in Example 3 and Example 4 are incomplete queries as the query in Example 3 doesn't contain start and end dates of the report and the query in Example 4 doesn't contain the conversation topic. The query in Example 5 contains both the start date and the end date but violates the domain constraint that the end date must be after the start date.

3.2 Intermediate Representation to Materialized Representation

An intermediate representation captures the content of the user query using formats and predicates that are closer to those of typical user utterances. In general, user queries cannot be expected to be formulated using the same vocabulary and format as the arguments of the APIs. In this step, we infer additional information as well as bridge the syntactic and semantic gaps. We accomplish this by using an ASP solver, with the intermediate representation and domain rules as inputs. For our current implementation we use Clingo [6] python package[3] as the ASP solver.

Below a snippet of the domain rules for our dataset (see Appendix B.2 for all domain rules). Note that even though the domain rules needed for our current dataset are rather simple and few in number, our framework of first translating the query to an intermediate representation in ASP allows us to plug-in a large number of complex rules if needed.

```
goal(X, profit_loss_report) :- _goal(X, goal_1).
start_date(X, Y, date) :- goal(X, profit_loss_report), _report_period(X, (Y,_)).
end_date(X, Y, date) :- goal(X, profit_loss_report), _report_period(X, (_,Y)).

goal(X, contact_us) :- _goal(X, goal_4).
contact_topic(X, Y, string) :- goal(X, contact_us), _contact_topic(X, Y).
contact_channel(X, Y, string) :- goal(X, contact_us), _contact_channel(X, Y).
...
error("end date must be after start date") :- start_date(X, D1, date),
    end_date(X, D2, date), false == @lte_dates(D1, D2).
```

[3] https://pypi.org/project/clingo/

The first rule translates the goal type to the type used in the vocabulary of the domain PDDL. The second and third rules infer `start_date` and `end_date` from the the user provided report_period. These rules also add the data types `date`, `string` for the values to facilitate the planning in the later step. The last rule infers an error when the end date is before the start date. In general, this technique allows us to generate error messages for complex constraint violations using ASP. For our example queries in Section 3.1, the ASP solver returns below materialized representations after applying the domain rules on the intermediate representations of the queries.

Materialized representation for Example 1:
```
goal(x, expense_spend_report).
start_date(x, "01/01/2023", date).
end_date(x, "03/31/2023", date).
```

Materialized representation for Example 2:
```
goal(x, profit_loss_report).
start_date(x, "07/01/2024", date).
end_date(x, "09/30/2024", date).
goal(y, contact_us).
contact_topic(y, x, string).
contact_channel(y, "phone", string).
```

Materialized representation for Example 3:
```
goal(x, profit_loss_report).
```

Materialized representation for Example 4:
```
goal(x, contact_us).
contact_channel(x, "chat", string).
```

Materialized representation for Example 5:
```
goal(x, expense_spend_report).
start_date(x, "07/01/2024", date).
end_date(x, "01/31/2024", date).
error("start date is after end date.").
```

Note that the materialized representation of Example 5 contains an error atom because the end date is before the start date. In such cases, we do not continue with task PDDL generation and send the error back to the user (see also Figure 2).

4 Orchestrate APIs using Planner

4.1 Task PDDL Generation

A materialized representation contains all user provided information in the target terminology and format. The next and the last step is to generate a plan. In order to be able to do that, we need to convert the materialized representation to a PDDL representation (task PDDL).

Figure 4 illustrates this process using Example 1. Every goal x becomes a *var* and every goal type t becomes a *var_type*. For each goal x of type t, (a) add (*has_type* x t) to the init section, (b) for each argument a of t and predicate p, a var x_a is added to the objects, (*has_type* x_a t) is added to init, (p x x_a) is added to goal, and if x_a has a value v, then (*has_value* x_a v) is added to init. Refer to Appendix C.1 for the complete algorithm for generating materialized representation to task PDDL. The output of the algorithm, the task PDDL for Example 1 is shown on the right side Figure 4. Refer to Appendix C.2 for the task PDDLs of other example queries.

4.2 Plan generation

Once the task PDDL is generated, all we need to do is to call a PDDL planner with the task PDDL and the domain PDDL. In our implementation we use the Fast Downward Planner [4] [12] with configuration parameters *alias = lama* and *search-time-limit = 1*. In other implementations, where compatibility to PDDL may not be important, one may also choose an appropriate ASP based planner [24].

[4] https://www.fast-downward.org/HomePage

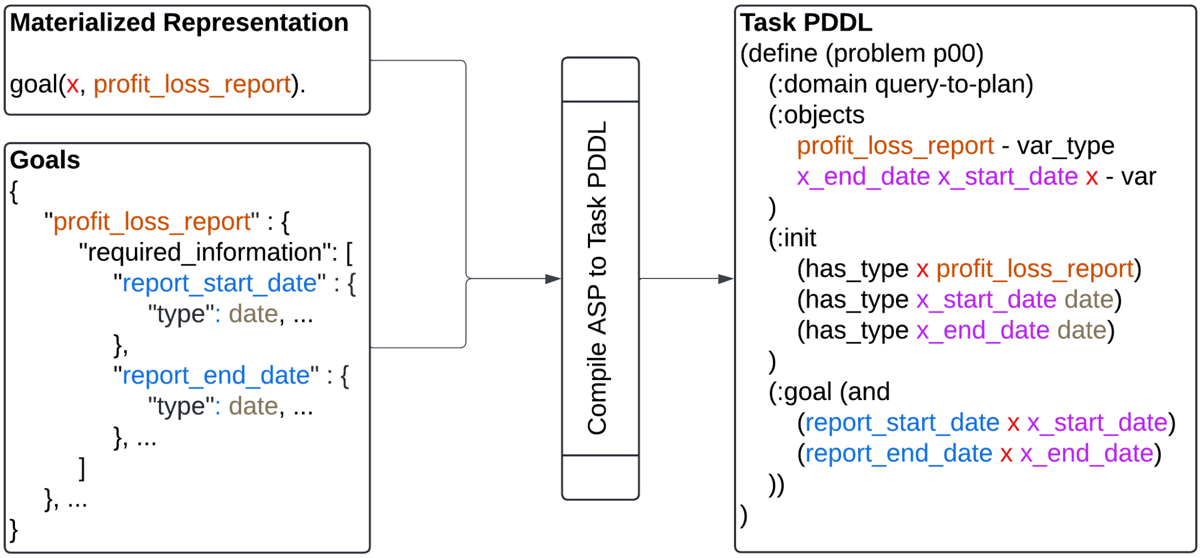


Figure 4: Materialized representation to task PDDL for the query *Profit and loss report*.

Using an external classical AI planner has several benefits such as: • *Scalability*: AI planners scale well wrt number of APIs as long as the functionality of APIs can be defined in terms of (Inputs, Outputs, Preconditions, Effects) with logical formulas. • *Support for interaction*: In case of incomplete queries the generated plan includes calls to get_info_api API for gathering information from user • *Optimality*: APIs can be assigned a cost; Planner computes an optimal plan wrt the cost function. • *Graceful failure*: For out of domain queries planner won't generate a plan rather than hallucinating.

For the example query *Show me 2023 Q1 detailed expense report*, the planner generates the plan:

```
Step 1. x_start_date = "01/01/2023";
Step 2. x_end_date = "03/31/2023";
Step 3. x = expense_spend_api(x_start_date, x_end_date);
```

For the example query *Provide me with the profit and loss statement for the previous quarter and then put me on a phone call with a representative to discuss it*, the planner generates:

```
Step 1. x_start_date = "07/01/2024";
Step 2. x_end_date = "09/30/2024";
Step 3. y_contact_channel = "phone";
Step 4. x = profit_loss_api(x_start_date, x_end_date);
Step 5. y = contact_us_api(x, y_contact_channel);
```

Note that the the contact topic is bound to the generated profit and loss report `x`.

For the example query *I want to chat with a representative*, the planner generates:

```
Step 1. x_contact_topic = get_info_api("contact topic", date);
Step 2. x_contact_channel = "chat";
Step 3. x = contact_us_api(x_contact_topic, x_contact_channel);
```

For the example query *Profit and loss report*, the planner generates:

```
Step 1. x_start_date = get_info_api("start date", date);
Step 2. x_end_date = get_info_api("end date", date);
Step 3. x = profit_loss_api(x_start_date, x_end_date);
```

5 Experiments

In this section we present the evaluation results of our approach on a generated dataset containing natural language user queries related to various topics such as generation of profit & loss reports, invoice creation, and how-to help requests.

5.1 Dataset Generation

The initial step in the dataset generation process involves using GPT-4 to generate user queries that represent single goal tasks executable via a subset of the APIs described in Section 2. GPT-4 is prompted with instructions and in-context examples to guide the generation process and ensure that the resulting queries align with the requirements of the API. Refer to Appendix D.1 for an example LLM prompt for dataset generation.

We use the same process to create more complex multi-goal queries simulating a real-world scenario where a user might seek to perform a series of actions in a single request. For example, "*Can I see my profit and loss statement from March to May 2023? I would like to discuss my profits further over chat.*". GPT-4 is prompted to generate coherent sequences where the output of one goal execution would become the input of another (multi-goal with dataflow), and complex queries which required multiple APIs to be executed independently (multi-goal without dataflow).

Once a sufficient number of single and multi-goal queries are generated, we first manually select queries that are representative of real user queries. Then, we manually annotate the selected queries with the ground truth values for the APIs and entities as their arguments. Refer to Appendix D.2 for some sample data in the dataset.

5.2 Results and Analysis

We consider a query as successfully processed iff the generated plan for answering the query contains all the ground truth APIs with correct entities as their arguments. In particular, the `get_info_api` calls correspond to missing entity values in incomplete queries. This allows us to also measure the success rate of incomplete queries where the planner should generate `get_info_api` actions for missing entities instead of the LLM hallucinating on entity values not present in the query. In our evaluation, a processed query is either correct or wrong, and never fractionally correct.

Table 1 and Table 2 present the average success rate (with a variance of 1.0) of our system over five runs on single goal and multi-goal queries respectively. The rows denote the different types of queries in our dataset. Columns 2 and 7 denoted by # represent the number of complete and incomplete queries respectively. In case of single goal queries, we report success rate for each goal type. In case of multi-goal queries, we distinguish between queries with 2 goals and 3 goals with or without dataflow. A query contains at least one goal and zero or more entities as arguments of the goals. The success rates reported in Table 1 and Table 2 are at most equal to the smaller of API orchestration success rate and entity values extraction success rate of the respective classes. See Appendix D.4 for API orchestration success rates and entity values extraction success rates.

We compare our method to a baseline where an LLM alone extracts the goals and entities in a query and performs orchestration of APIs. The baseline utilizes function calling method from [20] where APIs represented as function descriptions are used by the LLM to translate natural language query into function calls. Refer to Appendix D.3 for the LLM prompt used for the baseline approach. In our experiments, we observe that our approach significantly outperforms the baseline in most cases for single goal queries.

Table 1: Success rate % of our approach compared with a baseline of end-to-end LLM based approach on single goal queries

		Complete Queries					**Incomplete Queries**			
	#	**GPT-4**		**GPT3.5**		**#**	**GPT4**		**GPT3.5**	
		Base-line	Our Ap-proach	Base-line	Our Ap-proach		Base-line	Our Ap-proach	Base-line	Our Ap-proach
profit & loss report	70	22.86	**98.57**	81.43	**100**	2	0	**100**	0	**100**
expense report	42	23.81	**100**	90.48	**100**	0	-	-	-	-
invoice sales report	33	54.55	**90.91**	84.85	**93.94**	12	0	**100**	0	**91.67**
charge lookup	33	81.82	**100**	**96.97**	93.94	5	40.00	**100**	0	**100**
how-to help	60	68.33	**98.33**	68.33	**90.00**	0	-	-	-	-
contact us request	10	40.00	**100**	70.00	**100**	47	0	**91.49**	0	**85.11**
financial advice	100	81.00	**94.00**	94.00	**97.00**	0	-	-	-	-
create invoice	40	57.50	**100**	**100**	**100**	20	0	**100**	0	**100**
update customer	3	0	**100**	**100**	**100**	30	6.67	**100**	6.67	**100**

For complete queries, the baseline approach often fails to detect the correct goal or extract the entities in a query correctly. The former is mainly due to overlap in the API functionalities and thus the goals, e.g., there are three report generating APIs. The latter is due to large variation in expressing the same entity value. In addition, the baseline approach performs poorly on incomplete queries. In particular, the baseline approach with GPT-4 asks unnecessary clarification questions in case of complete queries and both GPT-4 and GPT-3.5 hallucinate on missing entity values in case of incomplete queries. We also observe that our approach can handle multi-goal complete and incomplete queries with high success rate while the baseline completely fails to orchestrate these queries correctly.

Overall, the increase in success rate in our approach can be attributed to the use of an LLM only for translating a user query coupled with the use of deterministic tools such as a logical reasoner and a planner for inferring additional information and generating a plan respectively. In particular, using an LLM to translate to an intermediate representation that is closer to the user query increases the translation accuracy as well as minimizes the hallucination. Furthermore, using a logical reasoner facilitates accurate mapping to target schema with the help of ASP rules even in complex domains where an LLM would often generate incorrect inferences. Similarly, using an external planner computes only feasible plans. In case of single goal complete queries, the increase in success rate is due to the use of intermediate representation and reasoning, and the planner doesn't add any additional value as the materialized representation itself can be seen as an equivalent to a plan. In case of single goal or multi-goal incomplete queries as well multi-goal complete queries with dataflow, the increase in the success rate is due to use of intermediate representation, logical reasoning, and the planner.

Our approach requires per query one LLM call, one ASP solver call, and one planner call. The total execution time for processing one query in case of GPT-4 is 3–5 seconds and 0.5–1 seconds in case of GPT-3.5. In both cases over 99% of total time is consumed by the LLM call(s) in the translation

Table 2: Success rate % of our approach compared with a baseline of end-to-end LLM based approach on multi-goal queries.

	#	**Complete Queries**				#	**Incomplete Queries**			
		GPT-4		**GPT-3.5**			**GPT-4**		**GPT-3.5**	
		Baseline	Our Approach	Baseline	Our Approach		Baseline	Our Approach	Baseline	Our Approach
2 APIs w/o dataflow	15	0	**100**	0	**100**	10	0	**100**	0	**100**
2 APIs with dataflow	20	0	**90**	0	**70**	10	0	**80**	0	**60**
3 APIs with dataflow	4	0	**100**	0	**75**	16	0	**75**	0	**62.50**

step. Note that our LLM response times are measured in a setup with shared resources across all LLM projects within our organization. We believe that the latency will be significantly lower with dedicated LLM access.

6 Conclusion

In this paper, we studied the problem of answering incomplete user queries in presence of APIs. To the best of our knowledge, ours is the first approach to address this problem. Our approach introduces a novel combination of LLMs, logical reasoning, and classical AI planning to support queries that can be complete or incomplete requiring only one API or an orchestration of multiple APIs. Furthermore, our approach supports queries of different kinds such as information seeking queries, how-to queries, and state changing queries. Our evaluation results show that our approach achieves high success rate (over 95% in most cases including 100% in some cases). Our approach is generic in the sense that it doesn't depend on a particular set of APIs but allows API specifications to be plugged in. The significant success rate improvement as compared to a pure LLM based baseline can be attributed to the use of interpretable intermediate representation, logical reasoning, and classical AI planning.

Our approach has a few limitations which we plan to address in our future work. Currently, we send the metadata for all supported goals of the domain to an LLM as part of the prompt. This technique can overshoot the LLM token limit in cases where there are a large number of possible goals in the domain. Currently, our approach only supports queries but not user's soft preferences. One way to address this gap, at least for some types of user preferences, could be to translate them to a cost function which AI planners can directly support. Lastly, the use of AI planner requires the APIs be specified with accurate IOPE specifications in PDDL which may not be applicable for all APIs or difficult to create for APIs with complex functionality.

References

[1] Sudhir Agarwal & Anu Sreepathy (2024): *TIC: Translate-Infer-Compile for accurate "text to plan" using LLMs and Logical Representations*. In: *Proceedings of the 18th International Conference on Neural-Symbolic Learning and Reasoning, Residència d'Investigadors Barcelona, Spain, September 9-12, 2024*, LNCS/LNAI, Springer. Available at `https://arxiv.org/abs/2402.06608`.

[2] Gerhard Brewka, Thomas Eiter & Miroslaw Truszczynski (2011): *Answer set programming at a glance*. *Commun. ACM* 54(12), pp. 92–103. Available at `https://doi.org/10.1145/2043174.2043195`.

[3] Harrison Chase (2022): *LangChain*. Available at `https://github.com/langchain-ai/langchain`.

[4] Luciano Floridi & Massimo Chiriatti (2020): *GPT-3: Its Nature, Scope, Limits, and Consequences*. *Minds Mach.* 30(4), pp. 681–694. Available at `https://doi.org/10.1007/s11023-020-09548-1`.

[5] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding & Jingren Zhou (2024): *Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation*. *Proc. VLDB Endow.* 17(5), pp. 1132–1145, doi:10.14778/3641204.3641221. Available at `https://www.vldb.org/pvldb/vol17/p1132-gao.pdf`.

[6] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2019): *Multi-shot ASP solving with clingo*. *Theory Pract. Log. Program.* 19(1), pp. 27–82. Available at `https://doi.org/10.1017/S1471068418000054`.

[7] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In: *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, MIT Press, pp. 1070–1080.

[8] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld & D. Wilkins (1998): *PDDL—The Planning Domain Definition Language*. Available at `https://www.cs.cmu.edu/~mmv/planning/readings/98aips-PDDL.pdf`.

[9] Malik Ghallab, Dana S. Nau & Paolo Traverso (2004): *Automated planning - theory and practice*. Elsevier.

[10] Lin Guan, Karthik Valmeekam, Sarath Sreedharan & Subbarao Kambhampati (2023): *Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning*. 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*.

[11] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni & Christian Muise (2019): *An Introduction to the Planning Domain Definition Language*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, doi:10.1007/978-3-031-01584-7.

[12] M. Helmert (2006): *The Fast Downward Planning System*. *Journal of Artificial Intelligence Research* 26, p. 191–246, doi:10.1613/jair.1705.

[13] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman & Brian Ichter (2022): *Inner Monologue: Embodied Reasoning through Planning with Language Models*. arXiv:2207.05608.

[14] Brian Ichter & ... (2022): *Do As I Can, Not As I Say: Grounding Language in Robotic Affordances*. In: *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, *Proceedings of Machine Learning Research* 205, PMLR, pp. 287–318. Available at `https://proceedings.mlr.press/v205/ichter23a.html`.

[15] Jack Kelly, Alex Calderwood, Noah Wardrip-Fruin & Michael Mateas (2023): *There and Back Again: Extracting Formal Domains for Controllable Neurosymbolic Story Authoring*. In: *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, October 08-12, 2023, Salt Lake City, UT, USA*, AAAI Press, pp. 64–74. Available at `https://doi.org/10.1609/aiide.v19i1.27502`.

[16] Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao & Hangyu Mao (2024): *PET-SQL: A Prompt-enhanced Two-stage Text-to-SQL Framework with Cross-consistency*. arXiv:2403.09732.

[17] Vladimir Lifschitz (2008): *What Is Answer Set Programming?* In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, AAAI Press, pp. 1594–1597. Available at `http://www.aaai.org/Library/AAAI/2008/aaai08-270.php`.

[18] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas & Peter Stone (2023): *LLM+P: Empowering Large Language Models with Optimal Planning Proficiency*. arXiv:2304.11477.

[19] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu & Jianfeng Gao (2023): *Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models*. 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*.

[20] OpenAI (2024): *Function calling*. Available at `https://platform.openai.com/docs/guides/function-calling`.

[21] Shishir G. Patil, Tianjun Zhang, Xin Wang & Joseph E. Gonzalez (2023): *Gorilla: Large Language Model Connected with Massive APIs*. arXiv:2305.15334.

[22] Abhiramon Rajasekharan, Yankai Zeng, Parth Padalkar & Gopal Gupta (2023): *Reliable Natural Language Understanding with Large Language Models and Answer Set Programming*. In: *Proceedings 39th International Conference on Logic Programming, ICLP 2023, Imperial College London, UK, 9th July 2023 - 15th July 2023*, *EPTCS* 385, pp. 274–287. Available at `https://doi.org/10.4204/EPTCS.385.27`.

[23] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda & Thomas Scialom (2023): *Toolformer: Language Models Can Teach Themselves to Use Tools*. arXiv:2302.04761.

[24] Tran Cao Son, Enrico Pontelli, Marcello Balduccini & Torsten Schaub (2023): *Answer Set Planning: A Survey*. *Theory Pract. Log. Program.* 23(1), pp. 226–298. Available at `https://doi.org/10.1017/S1471068422000072`.

[25] Bing Wang, Yan Gao, Zhoujun Li & Jian-Guang Lou (2023): *Know What I don't Know: Handling Ambiguous and Unknown Questions for Text-to-SQL*. In: *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, Association for Computational Linguistics, pp. 5701–5714.

[26] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le & Denny Zhou (2022): *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho & A. Oh, editors: *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*. Available at `http://papers.nips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html`.

[27] Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao & Yu Su (2024): *TravelPlanner: A Benchmark for Real-World Planning with Language Agents*. arXiv:2402.01622.

[28] 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, Toronto, Canada, July 9-14, 2023*, Association for Computational Linguistics, pp. 5186–5219. Available at `https://doi.org/10.18653/v1/2023.findings-acl.321`.

[29] Eric Zelikman, Qian Huang, Gabriel Poesia, Noah D. Goodman & Nick Haber (2023): *Parsel: Algorithmic Reasoning with Language Models by Composing Decompositions*. 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*. Available at `http://papers.nips.cc/paper_files/paper/2023/hash/6445dd88ebb9a6a3afa0b126ad87fe41-Abstract-Conference.html`.

[30] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Marcin Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael S. Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke & Pete Florence (2023): *Socratic Models: Composing Zero-Shot Multimodal Reasoning with Language*. In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, OpenReview.net. Available at `https://openreview.net/forum?id=G2Q2Mh3avow`.

[31] Yankai Zeng, Abhiramon Rajasekharan, Parth Padalkar, Kinjal Basu, Joaquín Arias & Gopal Gupta (2024): *Automated Interactive Domain-Specific Conversational Agents that Understand Human Dialogs*. In: *Practical Aspects of Declarative Languages - 26th International Symposium, PADL 2024, London, UK, January 15-16, 2024, Proceedings*, *Lecture Notes in Computer Science* 14512, Springer, pp. 204–222, doi:10.1007/978-3-031-52038-9_13.

A Translation Prompt

A.1 Argument Types

```
arg_type_date_period = {'examples': {'nov 2023' : ('11/01/2023',
    '11/30/2023'), 'april 15 2022-june 30 2022' : ('04/15/2022',
    '06/30/2022'), 'fy21': ('01/01/2021','12/31/2021'), '1 Half 2023':
    ('01/01/2023','06/30/2023'), '6/22 to 7/22': ('06/01/2022',
    '07/31/2022'), '1Q23' : ('01/01/2023', '03/31/2023'),
    'Mar-Apr 2022': ('03/01/2022','04/30/2022'),
    'April end-June start 2023': ('04/30/2023', '06/01/2023')}}

arg_type_date = {'examples' : {'1 nov 2023' : '11/01/2023',
    '11th November \'18': '11/11/2018', 'april 15 2022' : '04/15/2022',
    '2/21/18': '02/21/2018'}}

arg_type_amount = {'examples': {'$2': '2.00', '$15.90': '15.99',
    '$4,500' : '4500.00', '$65': '65.00'}}

arg_type_qb_feature = {'possible_values': ['accounts payable',
    'add trips manually', 'bank statements', 'budget', 'capital', '
    categorization', 'certification', 'change business name',
    'connect to bank', 'deposits', 'depreciation',
    'import journal entries', 'inventory', 'melio', 'overtime',
    'payroll', 'purchase order', 'purchase orders', 'reclassify',
    'recover deleted account', 'reset account', 'record an expense',
    'reconciliation', 'shortcuts', 'timesheets', 'timesheets/payroll',
    'vendors', 'write off bad debt']}

arg_type_conversation_topic = {'possible_values': [
    'account', 'Accounts Payable', 'Accounts Receivable',
    'Accounting Software', 'Bank Reconciliation', 'Billing',
    'Bookkeeping', 'Budget', 'Budget Tracking and Forecasting',
    'Cash Flow Management', 'Financial Analysis', 'Financial Planning',
    'Financial Reporting', 'Fixed Assets', 'insurance',
    'Inventory Management', 'Invoicing', 'issue', 'order', 'password',
    'Payroll', 'product', 'Purchase Orders', 'questions',
    'Reconciliations', 'returns', 'technical', 'shipping',
    'service_plan', 'tax', 'Tax Filing', 'Vendor Management']}

arg_type_conversation_channel = {'possible_values': ['speak', 'talk',
    'connect', 'video', 'chat', 'phone']}

arg_type_invoice_detail = {'possible_values':['Construction Project',
    'Tutoring Services', 'Website Design', 'Car Repair',
    'Catering Services', 'Event Management', 'Graphic Design',
```

```
    'Photography Service', 'Marketing Campaign',
    'Business Consultation', 'Furniture Supplies', 'Cleaning Service',
    'Painting Service', 'IT Consultancy', 'Accounting Services',
    'Renovation Work', 'Gardening Service', 'Legal Consultation',
    'Transportation Service', 'Personal Training Services',
    'catering service', 'marathon coaching', 'construction project',
    'baking class', 'introduction tutorial', 'lawn service',
    'grooming service', 'violin lesson', 'pilates session',
    'IT project', 'yoga class', 'personal training',
    'marketing consultation', 'premium subscription',
    'legal consultation', 'bartending service', 'reiki session',
    'mobile application development', 'home renovation service',
    'furnace inspection', 'dancing lessons', 'car repair service',
    'freelance design work', 'piano lessons', 'cleaning service',
    'plumbing service', 'hairstyling', 'landscaping service',
    'catering service', 'logistics service',
    'personal fitness training', 'graphic design work',
    'babysitting service', 'real estate consultancy', 'SEO services',
    'web development work', 'tailoring service', 'carpentry work',
    'security service', 'digital marketing service']}

arg_type_given_name = {'examples' : ['John', 'Mary']}

arg_type_family_name = {'examples' : ['Smith', 'Fischer']}

arg_type_email = {'examples' : ['j.fischer@abc.com']}

arg_type_phone = {'examples' : ['987-654-3210']}
```

A.2 Domain Goals

```
[{'type': 'goal_1',
    'description': 'request for generating a report on one of
     [profit_and_loss, income, business_insights, figures,
     operating_income, report, revenue, earnings].',
    'required information': [{'name': 'report_period',
     'description': 'time period defined by start and end dates.
      consider leap year while generating feb dates.',
      'type': arg_type_date_period}],
    'examples': [{'Earnings Summary?': '_goal(x, goal_1, earnings).'},
 {'type': 'goal_2',
    'description': 'request for generating a report on one of [expense,
      spend, bills, operating_expense, spend_figures, spending_insight,
      spend_report, expense_report, business_insights].',
    'required information': [{'name': 'report_period',
```

```
      'description': 'time period defined by start and end dates.
      Consider leap year while generating feb dates.', 'type':
      arg_type_date_period}],
    'examples': [{'What was the total expense for the first quarter of
    2023?':  '_goal(x, goal_2, expense_report). _report_period(x
      ("01/01/2023","03/31/2023")).'}]},
{'type': 'goal_3',
    'description': 'request for generating a report on one of
     [earnings, pending_payments, due_accounts, invoices,
     invoice_report, sales, accrued_expense, financial_forecast,
     revenue, report].',
    'required information': [{'name': 'report_period',
     'description': 'time period defined by start and end dates.
     Consider leap year while generating feb dates.', 'type':
     arg_type_date_period}], 'examples': [{'Show me all the
     invoices generated between March 1, 2022, and June 1, 2022':
     '_goal(x, goal_3, invoices).
    _report_period(x, ("03/01/2022","06/01/2022")).'}]},
{'type': 'goal_4', 'description': 'request for one of [charge_lookup]',
    'required_information': [{'name': 'date_of_charge', 'description':
    'date of charge.', 'type': arg_type_date},{'name':
    'amount_of_charge', 'description': 'amount of charge.', 'type':
    arg_type_amount}],
    'examples': [{'Why am I being charged $30.00?': '_goal(x, goal_4,
    charge_lookup). _amount_of_charge(x, "30.00").'}]},
{'type': 'goal_5', 'description': 'request for instructions
    on accomplishing a task in quickbooks.','required_information': [
    {'name': 'help_topic', 'description': 'quickbooks product
    feature relevant for the task to be accomplished', 'type':
    arg_type_qb_feature}],
    'examples': [{'What is the process for approving and
    fulfilling purchase orders?': '_goal(x, goal_5).
    _help_topic(x, "purchase orders").'}]},
{'type': 'goal_6',
    'description': 'request for a conversation with a person on the
    best matching conversation topic using best matching
    conversation medium.', 'required information': [{'name':
    'contact_topic', 'description': 'topic of conversation.', 'type':
    arg_type_conversation_topic}, {'name': 'contact_channel',
    'description': 'explicitly mentioned medium of
    conversation.', 'type': arg_type_conversation_channel},],
    'examples': [ {'I have some questions about billing. Can I chat
    with an expert about it?': '_goal(x, goal_6, expert).
    _contact_topic(x, "Billing"). _contact_channel(x, "chat").'},
    {'Can I speak to a representative?': '_goal(x, goal_6,
```

```
    representative). _contact_channel(x, "speak").'},
    {'Can I talk to an expert? What is the best way?': '_goal(x,
    goal_6, expert). _contact_channel(x, "talk").'},
    {'Can I book a phone call with a call agent?':
    '_goal(x, goal_6, call_agent). _contact_channel(x, "phone").'},
    {'Could I please speak with someone who can answer my questions?':
    '_goal(x, goal_6, representative). _contact_channel(x, "speak").
    _contact_topic(x, "questions").'}]},
{'type': 'goal_7',
    'description': 'request for an advice about one of ["business
    analysis", "business comparison", "business recommendation",
    "personal finance", "business expense", "profit making"].',
    'required information': [],
      'examples': [{'Any advice for dealing with monthly recurring
      expenses?': '_goal(x, goal_7).'}, {'How does my liquidity compare
      to similar businesses?': '_goal(x, goal_7).'}]},
{'type': 'goal_8',
    'description': 'request for creating a new invoice for a given
    amount and invoice detail.',
    'required information': [{'name': 'invoice_amount', 'description':
    'amount of invoice.', 'type': arg_type_amount},
    {'name': 'invoice_detail', 'description': 'detail of invoice.',
    'type': arg_type_invoice_detail},],
    'examples' : [{'Invoice needed of $200 for grooming service':
    '_goal(x, goal_8, new_invoice). _invoice_amount(x, "200.00").
    _invoice_detail(x, "grooming service").'}]},
{'type': 'goal_9', 'description': 'request for updating a customer
    profile', 'required information': [{'name': 'customer_given_name',
    'description': 'customer given name in customer profile.', 'type':
    arg_type_given_name}, {'name': 'customer_family_name',
    'description': 'customer family name in customer profile.', 'type':
    arg_type_family_name}, {'name': 'customer_email',
    'description': 'customer email in customer profile.' ,'type':
    arg_type_email}, {'name': 'customer_phone', 'description':
    'customer phone in customer profile.', 'type': arg_type_phone}],
    'examples': [{'Can you add a new profile for Henry Davis?':
    '_goal(x, goal_9, customer_profile). _customer_given_name
    (x, "Henry"). _customer_family_name(x, "Davis").'}]
}
]
```

A.3 In-context Examples

```
arg_type_color = {'name': 'color', 'description': 'color of an object',
    'type': {'description': 'color of an object', 'possible_values':
    ['red', 'green', 'blue', 'yellow']}}

arg_type_shape = {'name': 'shape', 'description': 'shape of an object',
    'type': {'description': 'shape on an object', 'possible_values':
    ['large', 'big', 'small', 'medium']}}

ex_goal_types = [
    {'type': 'fruits_goods', 'description': 'request for report about
    one of [apple, orange, ball].', 'required information':
    [arg_type_color, arg_type_shape]},
]
```

```
Goals: <AS ABOVE>

Text: show me red apples.

Answer:
% --- begin ---
_goal(x, fruits_goods, apple).
_color(x, "red").
% --- end ---

Goals: <AS ABOVE>

Text: which large oranges are green.

Answer:
% --- begin ---
_goal(x, fruits_goods, orange).
_color(x, "green").
_shape(x, "large").
% --- end ---

Goals: Goals: <AS ABOVE>

Text: big blue balls.

Answer:
% --- begin ---
_goal(x, fruits_goods, ball).
_color(x, "blue").
_shape(x, "big").
```

```
% --- end ---

Goals: Goals: <AS ABOVE>

Text: red oranges

Answer:
% --- begin ---
_goal(x, fruits_goods, orange).
_color(x, "red").
% --- end ---

Goals: Goals: <AS ABOVE>

Text: list of small oranges that are yellow

Answer:
% --- begin ---
_goal(x, fruits_goods, orange).
_color(x, "yellow").
_shape(x, "small").
% --- end ---
```

B Domain Modeling

B.1 Domain PDDL

```
(define (domain gen-orch-planner)
    (:requirements :strips)
    (:types
        var - object
        var_type - object
    )
    (:predicates
        (report_start_date ?r - var ?t - var)
        (report_end_date ?r - var ?t - var)
        (charge_date ?r - var ?t - var)
        (charge_amount ?r - var ?t - var)
        (help_topic ?r - var ?t - var)
        (contact_us_topic ?r - var ?t - var)
        (contact_us_channel ?r - var ?t - var)
        (invoice_amount ?r - var ?t - var)
        (invoice_detail ?r - var ?t - var)
        (customer_given_name ?r - var ?t - var)
```

```
        (customer_family_name ?r - var ?t - var)
        (customer_email ?r - var ?t - var)
        (customer_phone ?r - var ?t - var)
        (has_type ?a - var ?t - var_type)
        (has_value ?a - var)
     )

(:action get_info_api
    :parameters (?in_var - var ?in_type - var_type)
    :precondition (and (has_type ?in_var ?in_type)
        (not (has_value ?in_var)))
    :effect (and (has_value ?in_var)))

(:action profit_loss_api
    :parameters (?in1 - var ?in2 - var ?out - var)
    :precondition (and (has_type ?in1 date) (has_value ?in1)
        (has_type ?in2 date) (has_value ?in2)
        (has_type ?out profit_loss_report) (not (has_value ?out)))
    :effect (and (report_start_date ?out ?in1)
        (report_end_date ?out ?in2) (has_value ?out)))

(:action expense_spend_api
    :parameters (?in1 - var ?in2 - var ?out - var)
    :precondition (and (has_type ?in1 date) (has_value ?in1)
        (has_type ?in2 date) (has_value ?in2)
        (has_type ?out expense_spend_report) (not (has_value ?out)))
    :effect (and (report_start_date ?out ?in1)
        (report_end_date ?out ?in2) (has_value ?out)))

(:action invoice_sales_api
    :parameters (?in1 - var ?in2 - var ?out - var)
    :precondition (and (has_type ?in1 date) (has_value ?in1)
        (has_type ?in2 date) (has_value ?in2)
        (has_type ?out invoice_sales_report) (not (has_value ?out)))
    :effect (and (report_start_date ?out ?in1)
        (report_end_date ?out ?in2) (has_value ?out)))

(:action charge_lookup_api
    :parameters (?in1 - var ?in2 - var ?out - var)
    :precondition (and (has_type ?in1 date) (has_value ?in1)
        (has_type ?in2 number) (has_value ?in2)
        (has_type ?out charge_lookup_report) (not (has_value ?out)))
    :effect (and (charge_date ?out ?in1) (charge_amount ?out ?in2)
        (has_value ?out)))
```

```
(:action help_api
    :parameters (?in1 - var ?out - var)
    :precondition (and (has_type ?in1 string) (has_value ?in1)
        (has_type ?out help) (not (has_value ?out)))
    :effect (and (help_topic ?out ?in1) (has_value ?out)))

(:action contact_us_api
    :parameters (?in1 - var ?in2 - var ?out - var)
    :precondition (and (has_type ?in1 contact_topic) (has_value ?in1)
        (has_type ?in2 contact_channel) (has_value ?in2)
        (has_type ?out contact) (not (has_value ?out)))
    :effect (and (contact_us_topic ?out ?in1)
        (contact_us_channel ?out ?in2) (has_value ?out)))

(:action create_invoice_api
    :parameters (?in1 - var ?in2 - var ?out - var)
    :precondition (and (has_type ?in1 number) (has_value ?in1)
        (has_type ?in2 string) (has_value ?in2) (has_type ?out invoice)
        (not (has_value ?out)))
    :effect (and (invoice_amount ?out ?in1) (invoice_detail ?out ?in2)
        (has_value ?out)))

(:action update_customer_api
    :parameters (?in1 - var ?out - var)
    :precondition (and
        (has_type ?in1 customer_given_name) (has_value ?in1)
        (has_type ?in2 customer_family_name) (has_value ?in2)
        (has_type ?in3 customer_email) (has_value ?in3)
        (has_type ?in4 customer_phone) (has_value ?in4)
        (has_type ?out customer))
    :effect (and (customer_given_name ?out ?in1)
        (customer_family_name ?out ?in2) (customer_email ?out ?in3)
        (customer_phone ?out ?in4) (has_value ?out)))
)
```

B.2 Domain Rules

```
goal(X, profit_loss_report) :- _goal(X, goal_1, _).
start_date(X, Y1, date) :- goal(X, profit_loss_report),
    _report_period(X, (Y1, Y2)).
end_date(X, Y2, date) :- goal(X, profit_loss_report),
    _report_period(X, (Y1, Y2)).

goal(X, expense_spend_report) :- _goal(X, goal_2, _).
start_date(X, Y1, date) :- goal(X, expense_spend_report),
```

```
    _report_period(X, (Y1, Y2)).
end_date(X, Y2, date) :- goal(X, expense_spend_report),
    _report_period(X, (Y1, Y2)).

goal(X, invoices_sales_report) :- _goal(X, goal_3, _).
start_date(X, Y1, date) :- goal(X, invoices_sales_report),
    _report_period(X, (Y1, Y2)).
end_date(X, Y2, date) :- goal(X, invoices_sales_report),
    _report_period(X, (Y1, Y2)).

goal(X, charge_lookup) :- _goal(X, goal_4, _).
charge_date(X, Y, date) :- goal(X, charge_lookup),
    _date_of_charge(X, Y).
charge_amount(X, Y, number) :- goal(X, charge_lookup),
    _amount_of_charge(X, Y).

goal(X, helpgpt) :- _goal(X, goal_5).
help_topic(X, Y, string):- _help_topic(X, Y).

goal(X, contact_us) :- _goal(X, goal_6, _).
contact_topic(X, Y, fuzzy_string) :- goal(X, contact_us),
    _contact_topic(X, Y).
contact_channel(X, Y, fuzzy_string) :- goal(X, contact_us),
    _contact_channel(X, Y), Y == "video".
contact_channel(X, Y, fuzzy_string) :- goal(X, contact_us),
    _contact_channel(X, Y), Y == "chat".
contact_channel(X, Y, fuzzy_string) :- goal(X, contact_us),
    _contact_channel(X, Y), Y == "phone".

goal(X, advice) :- _goal(X, goal_7).

goal(X, create_invoice) :- _goal(X, goal_8, new_invoice).
invoice_amount(X, Y, number) :- goal(X, create_invoice),
    _invoice_amount(X, Y).
invoice_detail(X, Y, fuzzy_string) :- goal(X, create_invoice),
    _invoice_detail(X, Y).

goal(X, update_customer) :- _goal(X, goal_9, customer_profile).
customer_given_name(X, Y, string) :- goal(X, update_customer),
    _customer_given_name(X, Y).
customer_family_name(X, Y, string) :- goal(X, update_customer),
    _customer_family_name(X, Y).
customer_email(X, Y, string) :- goal(X, update_customer),
    _customer_email(X, Y).
customer_phone(X, Y, string) :- goal(X, update_customer),
```

```
    _customer_phone(X, Y).

error("end date must be after start date") :- start_date(X, D1, date),
    end_date(X, D2, date), false == @lte_dates(D1, D2).
...
```

C Query ASP to Task PDDL

C.1 Query ASP to Task PDDL Algorithm

str_objects ← ""
str_init ← ""
str_goal ← ""
Initialize objects ← $\{var\_type : \emptyset, var : \emptyset\}$. Initialize all_vars ← $\emptyset$.
for each goal in goals **do**
  Add goal to objects[var_type].
  for each var in goals[goal] **do**
    add goals[goal][var][type] to objects[var_type]
    add var to all_vars
for each atom in the ASP model **do**
  goal_var, goal_name ← atom.arguments[0], atom.arguments[1]
  if atom.name == "goal" **then**
    Add goal_var to objects[var]
    str_init += "(has_type " + goal_var + " " + goal_name + ")"
    for each arg of goals[goal_name] **do**
      arg_var ← goal_var+"_"+arg
      arg_type ← goals[goal_name][arg]["type"]
      pred_name ← goals[goal_name][arg]["predicate"]
      Add arg_var to $objects[var]$
      str_init += "(has_type " + arg_var + " " + arg_type + ")"
      str_goal += "(" + pred_name + " " + goal_var + " " + arg_var + ")"
  else if atom.name in all_vars **then**
    goal_var ← atom.arguments[0].name
    var ← goal_var + "_" + atom.name
    str_goal += "(" + atom.name + " " + goal_var + " " + var + ")"
str_objects += " ".join(objects[var]) + " - " + var
str_objects += " ".join(objects[var_type]) + " - " + var_type

C.2 Example Task PDDLs

```
(define (problem example1)
    (:domain query-to-plan)
    (:objects
        profit_loss_report - var_type
        x_end_date x_start_date x - var
```

```
    )
    (:init
        (has_type x profit_loss_report)
        (has_type x_start_date date)
        (has_type x_end_date date)
    )
    (:goal (and
        (report_start_date x x_start_date)
        (report_end_date x x_end_date)
    ))
)
```

```
(define (problem example2)
    (:domain query-to-plan)
    (:objects
        expense_spend_report - var_type
        x_end_date x_start_date x - var
    )
    (:init
        (has_type x profit_loss_report)
        (has_type x_start_date date)
        (has_value x_start_date "01/01/2023")
        (has_type x_end_date date)
        (has_value x_end_date "03/31/2023")
    )
    (:goal (and
        (report_start_date x x_start_date)
        (report_end_date x x_end_date)
    ))
)
```

```
(define (problem example3)
    (:domain query-to-plan)
    (:objects
        contact_us - var_type
        x_topic x_channel x - var
    )
    (:init
        (has_type x contact_us)
        (has_type x_topic string)
        (has_type x_channel string)
        (has_value x_channel "chat")
    )
    (:goal (and
        (contact_us_topic x x_topic)
        (contact_us_channel x x_channel)
```

```
    ))
)
```

```
(define (problem example4)
    (:domain query-to-plan)
    (:objects
        contact date contact_channel contact_topic
        profit_loss_report - var_type
        y x_end_date y_contact_channel x_start_date
        y_contact_topic x - var
    )
    (:init
        (has_type y contact)
        (has_type y_contact_topic contact_topic)
        (has_type y_contact_channel contact_channel)
        (has_type x profit_loss_report)
        (has_type x_start_date date)
        (has_type x_end_date date)
        (has_value x_start_date)
        (value x_start_date "last quarter start")
        (has_value x_end_date)
        (value x_end_date "last quarter end")
        (has_value y_contact_channel)
        (value y_contact_channel "phone")
    )
    (:goal (and
        (contact_us_topic y y_contact_topic)
        (contact_us_channel y y_contact_channel)
        (report_start_date x x_start_date)
        (report_end_date x x_end_date)
        (contact_channel y y_contact_channel)
    ))
)
```

D Evaluation

D.1 LLM prompt for dataset generation

Example prompt for generating user query and entities related to expense report

```
"Write 20 questions that use the variables below. These questions will
be used to test entity extraction.
The variables are
startperiod: the start date for the period of the expense and spend
endperiod: the end date the user wants for the expense and spend ,
```

```
Your response should be in the format following these examples:
{""Question"": ""spending breakdown"",
""startperiod"": [],
""endperiod"":[]
}

{""Question"": ""what have i spent most on 2020"",
""startperiod"": 1/1/2020,
""endperiod"": 12/31/2021
}

{""Question"": ""top monthly expenses from april 1 to may 2023"",
""startperiod"": 04/1/2023,
""endperiod"": 05/31/2023
}

{""Question"": ""top spending categories from 1/1/24 to 2/1/24"",
""startperiod"": 1/1/24,
""endperiod"": 2/1/24
}
"
```

D.2 Samples from the dataset

Query	gt_API	gt_entity1	gt_value1	gt_entity2	gt_value2
Q1 2023 P&L review?	profit_loss	startperiod	1/1/23	endperiod	3/31/23
Why was I charged $75?	charge_-lookup	dateofcharge	[]	amountofcharge	75

D.3 Baseline Prompt

```
{"role": "system", "content": """
Only use the functions you have been provided with. Do not assume or
hallucinate function parameters. If user has not provided, ask user for
required parameters.
Don't make assumptions about what values to plug into functions.
Ask for clarification if a user request is ambiguous.
"""},
{"role": "user", "content": query}
```

Here, functions are the APIs modelled as OpenAI function specifications and query refers to the user query of interest.

D.4 Evaluation Results

	Complete Queries					**Incomplete Queries**				
	#	**GPT-4**		**GPT-3.5**		**#**	**GPT-4**		**GPT-3.5**	
		Baseline	Our Approach	Baseline	Our Approach		Baseline	Our Approach	Baseline	Our Approach
profit & loss report	70	22.86	**98.57**	97.14	**100**	2	0	**100**	**100**	**100**
expense spend report	42	30.95	**100**	97.62	**100**	0	-	-	-	-
invoice sales report	33	63.64	**90.91**	87.88	**93.94**	12	16.67	**100**	66.67	**100**
charge lookup	33	81.82	**100**	**100**	96.97	5	40.00	**100**	**100**	**100**
how-to help	60	70.00	**98.33**	68.33	**90.00**	0	-	-	-	-
contact us request	10	40.00	**100**	80.00	**100**	47	14.89	**93.62**	44.68	**95.74**
financial advice	100	81.00	**94.90**	94.00	**97.00**	0	-	-	-	-
create invoice	40	60.00	**100**	**100**	**100**	20	0	**100**	**100**	**100**
update customer	3	0	**100**	100	**100**	30	6.67	**100**	**100**	**100**

Table 3: **API orchestration success rate %** of our approach compared with a baseline of end-to-end LLM based approach on **single goal queries**

	Complete Queries					Incomplete Queries				
	#	GPT-4		GPT-3.5		#	GPT-4		GPT-3.5	
		Baseline	Our Approach	Baseline	Our Approach		Baseline	Our Approach	Baseline	Our Approach
profit & loss report	70	22.86	**98.57**	81.43	**100**	2	0	**100**	0	**100**
expense spend report	42	23.81	**100**	90.48	**100**	0	-	-	-	-
invoice sales report	33	54.55	**96.97**	84.85	**96.97**	12	0	**100**	0	**91.67**
charge lookup	33	81.82	**100**	**96.97**	93.94	5	40.00	**100**	0	**100**
how-to help	60	68.33	**98.33**	68.33	**95.00**	0	-	-	-	-
contact us request	10	40.00	**100**	70.00	**100**	47	0	**97.87**	0	**87.23**
financial advice	100	81.00	**99.00**	94.00	**100**	0	-	-	-	-
create invoice	40	57.50	**100**	**100**	**100**	20	0	**100**	0	**100**
update customer	3	0	**100**	**100**	**100**	30	6.67	**100**	6.67	**100**

Table 4: **Entity extraction success rate %** of our approach compared with a baseline of end-to-end LLM based approach on **single goal queries**

	Complete Queries					Incomplete Queries				
	#	GPT-4		GPT-3.5		#	GPT-4		GPT-3.5	
		Baseline	Our Approach	Baseline	Our Approach		Baseline	Our Approach	Baseline	Our Approach
2 APIs w/o dataflow	15	0	**100**	0	**100**	10	0	**100**	0	**100**
2 APIs with dataflow	20	0	**100**	0	**80**	10	0	**80**	0	**80**
3 APIs with dataflow	4	0	**100**	0	**75**	16	0	**100**	0	**81.25**

Table 5: **API orchestration success rate %** of our approach compared with a baseline of end-to-end LLM based approach on **multi goal complete queries**

	Complete Queries					Incomplete Queries				
	#	GPT-4		GPT-3.5		#	GPT-4		GPT-3.5	
		Baseline	Our Approach	Baseline	Our Approach		Baseline	Our Approach	Baseline	Our Approach
2 APIs w/o dataflow	15	0	**100**	0	**100**	10	0	**100**	0	**100**
2 APIs with dataflow	20	0	**90.00**	0	**70**	10	0	**90**	0	**60**
3 APIs with dataflow	4	0	**100**	0	**75.00**	16	0	**75**	0	**62.50**

Table 6: **Entity extraction success rate %** of our approach compared with a baseline of end-to-end LLM based approach on **multi goal queries**

Logical Lease Litigation: Prolog and LLMs for Rental Law Compliance in New York

Sanskar Sehgal Yanhong A. Liu
Stony Brook University, Stony Brook, NY 11794, USA
{sasehgal, liu}@cs.stonybrook.edu

Abstract

Legal cases require careful logical reasoning following the laws, whereas interactions with non-technical users must be in natural language. As an application combining logical reasoning using Prolog and natural language processing using large language models (LLMs), this paper presents a novel approach and system, LogicLease, to automate the analysis of landlord-tenant legal cases in the state of New York.

LogicLease determines compliance with relevant legal requirements by analyzing case descriptions and citing all relevant laws. It leverages LLMs for information extraction and Prolog for legal reasoning. By separating information extraction from legal reasoning, LogicLease achieves greater transparency and control over the legal logic applied to each case. We evaluate the accuracy, efficiency, and robustness of LogicLease through a series of tests, achieving 100% accuracy and an average processing time of 2.57 seconds. LogicLease presents advantages over state-of-the-art LLM-based legal analysis systems by providing clear, step-by-step reasoning, citing specific laws, and distinguishing itself by its ability to avoid hallucinations—a common issue in LLMs.

1 Introduction

Rental law compliance matters significantly to all rental residents. According to [2], more than 122.8 million households in the United States are renters. [5] estimates that there are over 1.1 million cases of landlords evicting tenants every year, representing an increase of 75% since 2021. Furthermore, 60% of eviction case defendants in 2023 were women, and despite making up less than one-third of renters, nearly half of eviction case defendants in 2023 were Black. [4] states that over seven million Americans are evicted from their homes every year, nearly 40% (2.7 to 3.2 million) of which are children. Additionally, in the United States, there is no guaranteed right to legal counsel in eviction proceedings. [16] estimates that as many as 90% of tenants facing eviction go to court unrepresented, putting them at a significant disadvantage. Consequently, up to 75% of tenants end up losing their eviction cases.

The legal domain's emphasis on meticulous analysis and transparent thought processes makes it an ideal candidate for utilizing logic-based systems over black-box approaches. Logic-based systems excel in providing clear and transparent reasoning, which is highly valued in legal contexts. While large language models (LLMs) are increasingly being considered for automating legal analysis, they are prone to hallucinations, which can lead to incorrect legal interpretations.

This paper presents LogicLease, a novel system specifically designed to automate the analysis of landlord-tenant legal cases in the state of New York, providing a transparent and reliable alternative to black-box methods.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 59–68, doi:10.4204/EPTCS.416.4

Rental law compliance is crucial for the well-being of tenants, but navigating legal cases can be complex and time-consuming. With millions of eviction cases each year, many tenants face eviction without proper legal representation, resulting in a significant disadvantage.

LogicLease harnesses the strengths of Large Language Models (LLMs) for information extraction and Prolog for legal reasoning. It is designed to assess compliance with relevant legal requirements by analyzing case descriptions presented in natural language. LogicLease utilizes LLMs to parse the input, extracting essential details that can influence case outcomes. Subsequently, it employs a logic-based backend to generate clear, step-by-step reasoning and cite specific laws.

The accuracy, efficiency, and robustness of LogicLease were evaluated through a series of tests, demonstrating 100% accuracy and an average processing time of 2.57 seconds. By separating information extraction from legal reasoning, LogicLease ensures greater transparency and control over the legal logic applied to each case. Furthermore, LogicLease addresses the challenges of hallucinations and opacity common in LLMs, providing a reliable tool for landlord-tenant legal analysis in New York.

The rest of the paper is organized as follows. Section 2 describes the problem of reliable analysis of rental law compliance. In Section 3, we present the approach used in LogicLease for automating the analysis of landlord-tenant legal cases. Section 4 provides implementation details and evaluates the accuracy, efficiency, and robustness of LogicLease. Finally, Section 5 discusses related work and concludes the paper.

2 The need for legal compliance analysis

Millions of Americans face challenges in their homes, and those from marginalized communities are often disproportionately impacted by unfair housing practices. To address this gap, we developed LogicLease. LogicLease promises to provide free and highly accurate guidance on tenant-landlord issues. Our system is designed for simplicity, so anyone can navigate it, regardless of technical expertise. We prioritize accuracy to ensure users receive reliable information to confidently address their housing concerns.

When developing LogicLease, we established three primary requirements:

1. Natural Language Input: Users can describe their situation in plain English (or potentially another language). This could involve describing the issue (e.g., repairs not being made, rent increase concerns), relevant details (lease terms, dates), and any questions they have.

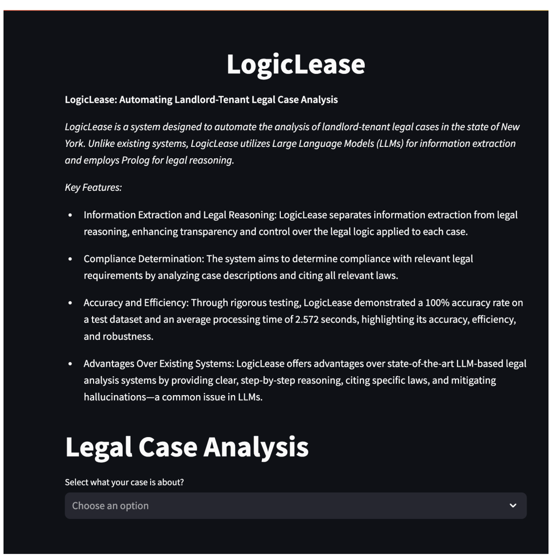
2. Rigorous Analysis for High Accuracy: The system uses LLMs to understand the user's situation and intent. It analyzes legal regulations, relevant case law, and best practices using the Prolog backend to provide accurate guidance. This might involve identifying the key legal issues involved, assessing the user's rights and responsibilities based on their location and lease agreement and considering potential solutions or next steps.

3. Natural Language Output: The system provides clear and actionable information tailored to the user's situation. This output is generated using a Prolog backend, where description of relevant laws is coded as strings associated with the rules. As the Prolog backend processes the rules and retrieve the specific law applicable to the case, it generates and prints the corresponding description to the output. This may include explanations of tenant rights or step-by-step guidance on how to address issues (e.g., contacting the landlord, or filing a complaint), presented in a way that is easy to understand, even for people with no legal background.

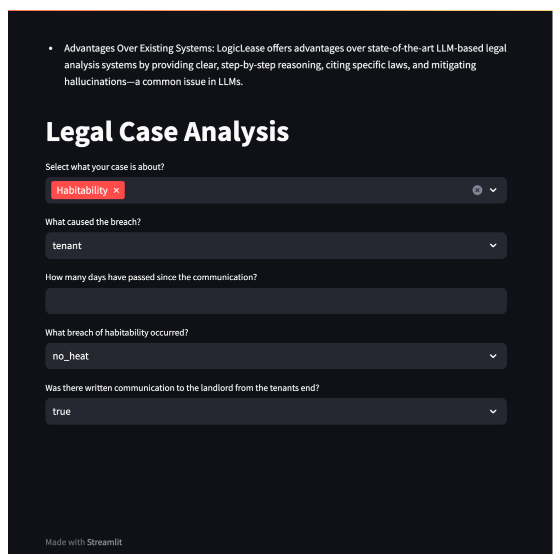
3 Combining logic programming and LLMs for legal analysis

LogicLease consists of four main components:

1. Driver Script: This component serves as the central coordinator, orchestrating the interaction between the other components. It takes a case description (lease agreement) as input and invokes the Natural Language Processing (NLP) and Prolog functionalities for analysis.
2. Natural Language Processing (NLP): LogicLease utilizes a large language model (LLM) out-of-the-box, without any additional training, to extract attribute-value pairs from the lease agreement. These pairs represent critical aspects of the case, such as whether the lease is signed or the duration of the rental period.
3. Prolog Knowledge Base: This component contains a set of Prolog rules that represent legal requirements for rentals, which we manually coded based on the New York Renters' Rights Handbook [12]. It contains an exhaustive set of rules about lease validity, rent stabilization, eviction, habitability, and more. LogicLease utilizes the attribute-value pairs extracted by the LLM as arguments for Prolog queries and uses the Prolog engine to evaluate compliance against these rules.
4. User Interface: The user interface of LogicLease provides users with an intuitive and user-friendly experience, enabling them to interact with the system seamlessly (Figure 1a). Users input the case description in natural language. Alternatively, they can also answer a series of dynamically generated questions related to the specific legal aspects of their case. These questions are designed to extract key attributes necessary for the legal analysis (Figure 1b). Users can provide answers using dropdown menus or text input fields. The user interface processes the information and invokes the legal analysis.



(a) User Interface of LogicLease



(b) Example of dynamically generated questions

Figure 1: LogicLease front-end interface

As shown in Figure 2, the design of LogicLease follows a structured process. Initially, the Driver Script receives a case description in natural language as input. Subsequently, it calls the API to the LLM, providing a system query that specifies the desired lease agreement aspects and the user query containing the actual case description. The API call sends the queries to the LLM, retrieving a response containing extracted attribute-value pairs. The Driver Script parses this response and stores the extracted attribute-value pairs in a dictionary. Following this, the Driver Script invokes Prolog functions passing the dictionary containing the extracted attribute-value pairs. These functions translate the dictionary into a Prolog query based on the pre-defined Prolog rules in the knowledge base. The Prolog engine then evaluates the query, determining compliance with the lease agreement requirements. Finally, the results of the compliance check are displayed by the Driver Script.

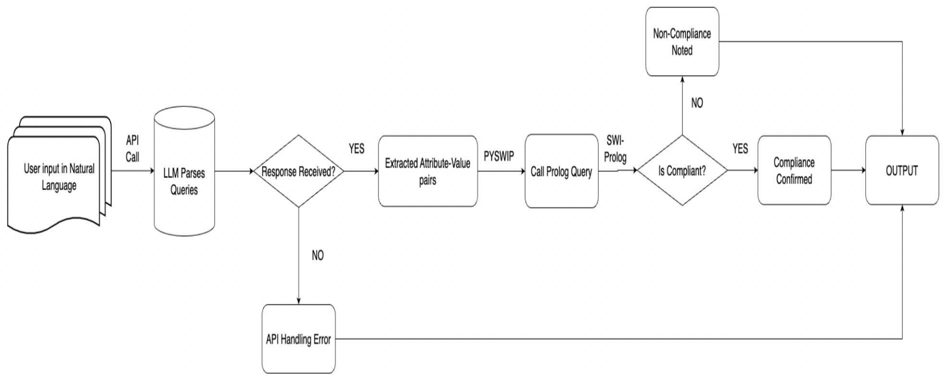


Figure 2: Design diagram of LogicLease

To illustrate the workflow of LogicLease and to facilitate a deeper understanding of the system, we provide an overview of the input processing, attribute-value pair extraction using an LLM, compliance checking with the Prolog backend system, and the final output.

User input of case in natural language. The input for a case is taken in natural language. An example is the following:

> In a rent-stabilized apartment in Albany, New York, David, a disabled tenant, faced eviction proceedings initiated by his landlord, Ms. Johnson, citing owner occupancy as the cause. Despite David's disability, he has been asked to vacate. The matter has not been presented before court, and hence does not have a court ruling yet.

This input is passed to the LLM for processing.

Attribute-value pairs extracted. The LLM parses the input text and outputs attribute-value pairs which are the core details of the case which influence the outcome. The extracted attribute-value pairs indicate that the eviction cause is ”owner occupancy,” there is no court ruling yet (”CourtRuling” is ”false”), and the tenant is in a protected category (”TenantCategory” is ”disabled”). These attribute-value pairs are stored in a dictionary in the following format.

EvictionCause : ”owner_occupancy”,
CourtRuling : ”false”,
Executioner : ”null”,
TenantCategory : ”disabled”,

Logic rules and queries in Prolog. As seen in Listing 1, the compliance with legal requirements in this specific case is determined by invoking the following Prolog predicate. Notably, the arguments precisely match the information extracted by the Large Language Model (LLM) from the case description.

```
eviction(EvictionCause, CourtRuling, Executioner, TenantCategory) :-
    eviction_law,
    (CourtRuling == true ->
        (
            eviction_legal(EvictionCause),
            writeln('All conditions satisfied, eviction is lawful.'),
            nl
        )
    ;
    (
        eviction_warrant_execution(Executioner),
        \+ overrides(TenantCategory, EvictionCause),
        writeln('All conditions satisfied, eviction is lawful.'),
        nl
    )
).
```

Listing 1: Prolog rule for eviction compliance checking

The arguments of the predicate are instantiated based on the extracted attribute-value pairs stored in the dictionary. The call to the Prolog predicate is structured as follows:

```
Prolog.query(eviction(owner_occupancy, false, null, disabled).)
```

The Prolog query evaluates the compliance of the landlord's actions with the relevant legal requirements.

Final output to user in natural language. The final output consists of a list of laws relevant to the case followed by a final judgment based on these laws. This transparent mechanism provides a clear rationale for each case. For our example, the output is:

1. Tenant with a lease is protected from eviction during the lease period if lease provisions and local laws are not violated.
2. Landlords must give formal notice before seeking legal possession of the apartment.
3. Eviction proceedings can be initiated for non-payment or significant lease violations.
4. Landlords of rent-regulated apartments may need DHCR approval for court proceedings.
5. Tenants should not ignore legal papers to avoid eviction.
6. Legal eviction requires court proceeding and judgment of possession.
7. Landlords cannot evict tenants unlawfully or by force.
8. Tenant evicted unlawfully can recover triple damages.
9. Additional rules protect certain groups from eviction.

Tenant is in protected category, eviction for owner occupancy unlawful.

The output confirms that the eviction for owner occupancy is unlawful given the tenant's protected category status and the absence of a court ruling.

4 Implementation and evaluation

We have developed a complete implementation of the LogicLease system, covering the full New York State landlord-tenant legal framework. We successfully applied our system to a series of 10 test cases, representing various scenarios encountered in landlord-tenant disputes. To evaluate the system's performance, we employ a multi-pronged approach, assessing its accuracy, efficiency, and robustness across various dimensions.

4.1 Implementation

LogicLease is implemented using a combination of Python and SWI-Prolog [21, 17]. Python is used to facilitate interactions among different components, utilizing libraries such as PYSWIP to establish communication between the Python script and the Prolog engine (SWIPL), while Streamlit is employed to create a user-friendly front-end interface. Additionally, llamaapi [10] is used to manage interactions with the Large Language Model (LLM) model LLaMA [18], while Python libraries like json and ast are employed for processing the output received from LLaMA.

The Prolog backend serves as the foundation for legal reasoning, with custom clauses defining the relevant laws and procedures. The implementation of the system followed a structured approach, ensuring modularity and ease of maintenance.

Notably, LogicLease incorporated defeasible logic [19, 20, 11] within the Prolog component, enabling the system to handle situations where one legal principle takes precedence over another under specific circumstances.

By separating information extraction (NLP) from legal reasoning (Prolog), LogicLease achieved greater transparency and control over the legal reasoning applied to each case. This approach is particularly important for legal professionals who require a clear understanding of the system's reasoning process, while ensuring there are no hallucinations or snowballing effects.

The total size is approximately 500 lines of Prolog code and an additional 400 lines of Python code.

All experiments and measurements were conducted on a macOS system featuring an Apple Silicon M2 processor, 8GB of RAM, and a 256GB SSD. The system was running macOS Monterey version 12.5, with Python 3.9.12, Prolog SWIPL 9.2.2, PYSWIP 0.2.11, llamaapi version 0.1.36, and Streamlit version 1.24.1.

4.2 Accuracy

To evaluate LogicLease's effectiveness, we manually compiled a dataset of lease litigation cases in New York. This dataset includes a mix of real-world cases (condensed for efficiency) and fictional scenarios we created. Due to limited API credits available for using the LLM, the system restricted the number of API calls made during the processing of legal cases. This limitation necessitated significant shortening of the text within the legal documents to fit within the allowed API usage. Despite these constraints, the resulting dataset of 10 cases effectively demonstrates the system's accuracy and usability in analyzing and responding to legal scenarios.

Encouragingly, in all ten cases tested, the LLM functioned effectively. It accurately extracted relevant details from each case description and successfully transferred this information to the Prolog backend system. The Prolog system, in turn, flawlessly interpreted the queries and delivered final verdicts on the legal issues presented. Although the dataset is currently small, the system's 100% accuracy on this dataset helps build trust in the system.

To evaluate the accuracy of the system's reasoning, we conducted human-in-the-loop evaluation. This involved manually reviewing the system's output for each case. Legal resources such as handbooks and online legal forums (e.g., on Reddit) were used to verify the system's determinations. This process helped identify potential biases in the LLM's interpretation of the case details or errors within the Prolog code's reasoning logic.

4.3 Efficiency

We measured the average processing time per case, including the LLM extraction and the logic-based compliance check. The processing times are shown in Table 1. The main bottleneck is the API call to the LLM Llama, which is not directly under our control. However, solutions such as caching frequently accessed legal information can be explored.

While the LLM API call (to Llama) currently represents the performance bottleneck compared to Prolog's minimal processing time, the good news is that the average case processing time remains user-friendly at only 2.572 seconds.

It also helps if the case description passed to Llama is as concise as possible. Not surprisingly, the time taken for Llama to respond is directly proportional to the length of the query string passed. In fact, the system exhibits a strong positive correlation (approx 68%) between the length of the query string and Llama's response time.

Test Case	Total Time	Prolog Running Time
1	2.693	0.00012
2	2.462	0.00018
3	2.412	0.00021
4	3.156	0.00120
5	3.324	0.00130
6	2.270	0.00090
7	2.155	0.00050
8	2.215	0.00042
9	2.551	0.00031
10	2.486	0.00010
Average	**2.572**	**0.00011**

Table 1: Running times, in seconds, for all test cases

4.4 Robustness

To highlight the superiority and robustness of our system compared to existing solutions, we present a comparison with state-of-the-art Large Language Models (LLMs) such as ChatGPT [13] and Gemini [3]. These LLMs appear to favor the landlord in most cases, as shown by their outputs for the running example shown in Figures 4 and 5. Neither of these LLMs reached the correct conclusion for this example

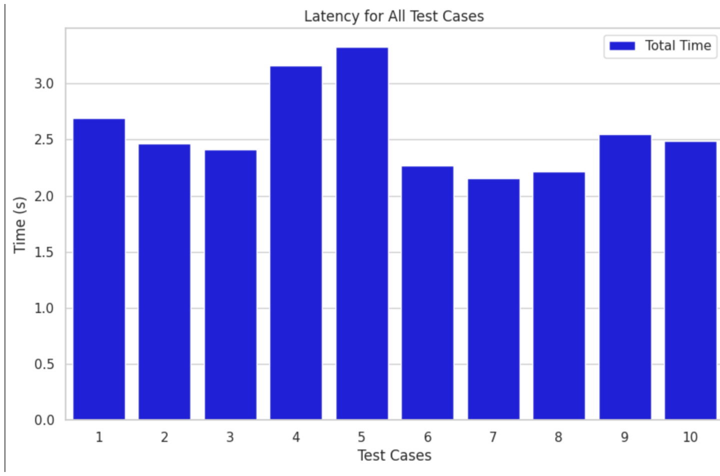


Figure 3: Total Time Taken for Each Test Case

case. According to New York law, the situation clearly favors the tenant if they live in a rent-stabilized apartment and belong to a protected category. For our test dataset, these LLMs did not reach the correct conclusion in three out of ten cases. This underscores the importance of our system and highlights the unreliability of LLMs for this application.

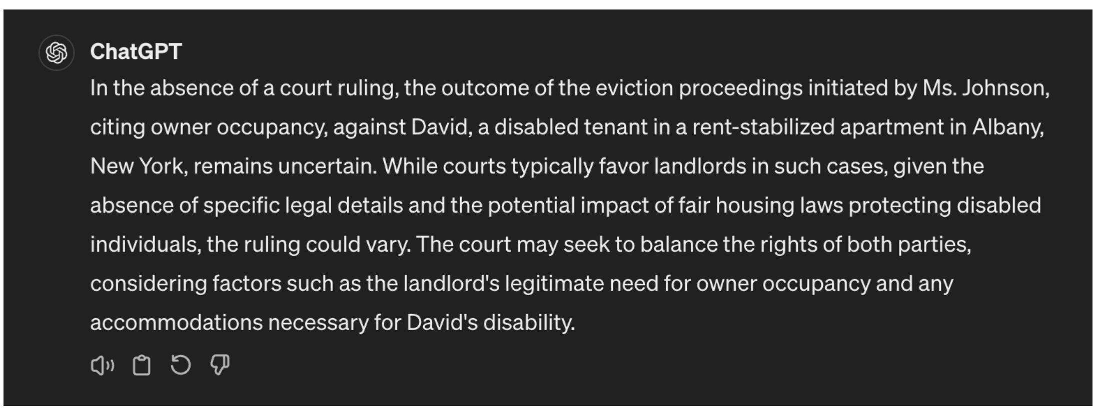


Figure 4: ChatGPT output

5 Related work and conclusion

Existing systems for legal analysis employ various techniques, including text classification, machine learning, and rule-based reasoning [1, 9, 6, 7, 8]. However, these approaches often have limitations in handling nuanced legal reasoning.

Text classification systems such as Legal-Document-Classifier [1] and LexNLP [9] can categorize legal documents based on keywords and named entities but lack the ability to perform comprehensive legal reasoning.

Machine learning-based systems like Kira [6] and LawGeex [7] can extract key terms and identify potential issues in contracts, and models like Lex Machina [8] can predict legal outcomes with some

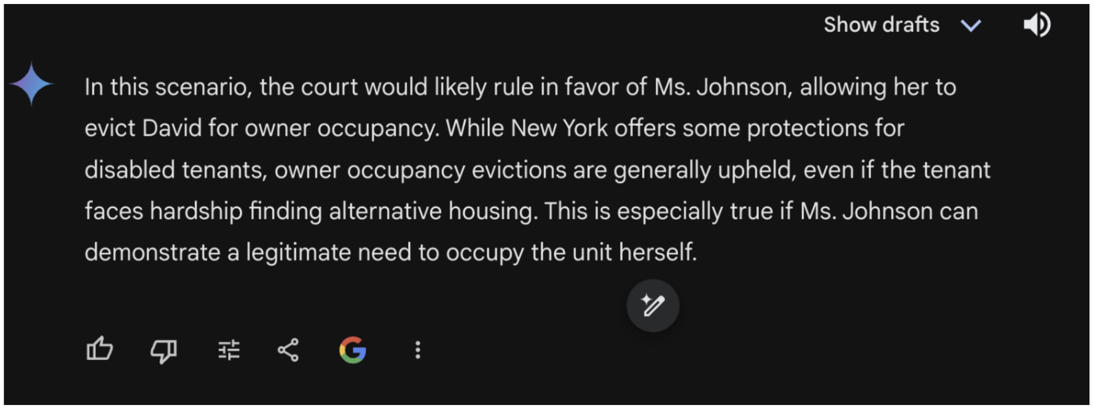


Figure 5: Gemini output

accuracy. However, these systems often operate as black boxes, raising concerns about transparency and fairness. They may also be limited by data quality and biases.

Rule-based legal reasoning systems like PROLEG [14, 15] offer support for judges in civil litigation by incorporating predefined rules and handling uncertainty. However, their complexity can pose challenges for users.

In contrast, we successfully developed a system for analyzing landlord-tenant disputes in New York State by leveraging Large Language Models (LLMs) for information extraction and Prolog for legal reasoning. Achieving high accuracy and efficiency, the system offers several advantages over existing LLM-based legal analysis systems.

By separating information extraction from legal reasoning, the system achieves greater transparency and control over the legal logic applied to each case. Additionally, the use of Prolog enables the implementation of defeasible logic [19, 20], allowing the system to handle nuanced legal reasoning, such as resolving conflicting legal principles and dealing with uncertain or incomplete information [11].

In conclusion, this work demonstrates the potential of combining Large Language Models (LLMs) and logic-based reasoning to create innovative tools for legal analysis. By addressing the limitations of existing approaches, LogicLease paves the way for more sophisticated and transparent systems in the field of legal technology.

Future work includes expanding the system's capabilities by employing techniques for caching frequently accessed legal information. Additionally, improving the testing process and adding more test cases will ensure better coverage and reliability of the system. Open-sourcing the code could encourage further development and broader adoption within the legal domain.

References

[1] Andrei-Marius Avram, Vasile Pais & Dan Tufis (2021): *PyEuroVoc: A Tool for Multilingual Legal Document Classification with EuroVoc Descriptors*. ArXiv preprint arXiv:2108.01139.

[2] Drew Desilver (2021): *As national eviction ban expires, a look at who rents and who owns in the US.* `https://www.pewresearch.org/short-reads/2021/08/02/as-national-eviction-ban-expires-a-look-at-who-rents-and-who-owns-in-the-u-s/`.

[3] Google (2024): *Gemini Large Language Model*. Large language model. Available at `https://gemini.google.com/app`.

[4] Nick Graetz, Carl Gershenson, Peter Hepburn, Sonya R Porter, Danielle H Sandler & Matthew Desmond (2023): *A comprehensive demographic profile of the US evicted population*. Proceedings of the National Academy of Sciences 120(41), doi:10.1073/pnas.2305860120.

[5] Peter Hepburn, Olivia Jin, Joe Fish, Emily Lemmerman, Anne Kat Alexander & Matthew Desmond (2022): *Preliminary analysis: Eviction filing patterns in 2021*. https://evictionlab.org/us-eviction-filing-patterns-2021/.

[6] KiraSystems (2015): *Kira*. https://kirasystems.com/.

[7] LawGeex (2014): *LawGeex*. https://www.lawgeex.com.

[8] Lemley Mark, Gregory George & Walker Joshua (2006): *Lex Machina*. https://lexmachina.com/.

[9] LexPredict (2024): *LexNLP by LexPredict*. https://github.com/LexPredict/lexpredict-lexnlp.

[10] llamaapi, Reis, Eduardo & Hu, Jesse (2023): *llamaapi-python*. https://github.com/llamaapi/llamaapi-python.

[11] Jason Morris (2024): *Rules as Code Demonstration in SWISH and s(CASP)*. https://swish.swi-prolog.org/p/r34v2.swinb. Accessed Mar. 17, 2024.

[12] New York State Attorney General (2024): *Tenants' Rights Guide*. https://ag.ny.gov/sites/default/files/tenants_rights.pdf.

[13] OpenAI (2023): *ChatGPT (Mar 14 version)*. Large language model. Available at https://chat.openai.com/chat.

[14] Ken Satoh, Kento Asai, Takamune Kogawa, Masahiro Kubota, Megumi Nakamura, Yoshiaki Nishigai, Kei Shirakawa & Chiaki Takano (2010): *PROLEG: an implementation of the presupposed ultimate fact theory of Japanese civil code by PROLOG technology*. In: *JSAI international symposium on artificial intelligence*, Springer, pp. 153–164, doi:10.1007/978-3-642-25655-4_14.

[15] Ken Satoh, Kento Asai, Takamune Kogawa, Masahiro Kubota, Megumi Nakamura, Yoshiaki Nishigai, Kei Shirakawa & Chiaki Takano (2011): *PROLEG: An Implementation of the Presupposed Ultimate Fact Theory of Japanese Civil Code by PROLOG Technology*. In Takashi Onada, Daisuke Bekki & Elin McCready, editors: *New Frontiers in Artificial Intelligence*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 153–164, doi:10.1007/978-3-642-25655-4_14.

[16] Heidi Schultheis & Caitlin Rooney (2019): *A right to counsel is a right to a fighting chance*. https://www.americanprogress.org/article/right-counsel-right-fighting-chance/.

[17] (2024): *SWI Prolog*. https://www.swi-prolog.org. Accessed Mar. 17, 2024.

[18] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar et al. (2023): *LlaMa: Open and efficient foundation language models*, doi:10.48550/arXiv.2302.13971. ArXiv preprint arXiv:2302.13971.

[19] Hui Wan, Benjamin Grosof, Michael Kifer, Paul Fodor & Senlin Liang (2009): *Logic programming with defaults and argumentation theories*. In: *Logic Programming: 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings 25*, Springer, pp. 432–448, doi:10.1007/978-3-642-02846-5_35.

[20] Hui Wan, Michael Kifer & Benjamin Grosof (2015): *Defeasibility in answer set programs with defaults and argumentation rules*. Semantic Web 6(1), pp. 81–98, doi:10.3233/SW-140140.

[21] Jan Wielemaker, Tom Schrijvers, Markus Triska & Torbjörn Lager (2012): *SWI-Prolog*. Theory and Practice of Logic Programming 12(1-2), pp. 67–96, doi:10.1017/S1471068411000494.

LP-LM: No Hallucinations in Question Answering with Logic Programming

Katherine Wu*
Cornell University
kaw324@cornell.edu

Yanhong A. Liu
Stony Brook University
liu@cs.stonybrook.edu

Large language models (LLMs) are able to generate human-like responses to user queries. However, LLMs exhibit inherent limitations, especially because they hallucinate. This paper introduces LP-LM, a system that grounds answers to questions in known facts contained in a knowledge base (KB), facilitated through semantic parsing in Prolog, and always produces answers that are reliable.

LP-LM generates a most probable constituency parse tree along with a corresponding Prolog term for an input question via Prolog definite clause grammar (DCG) parsing. The term is then executed against a KB of natural language sentences also represented as Prolog terms for question answering. By leveraging DCG and tabling, LP-LM runs in linear time in the size of input sentences for sufficiently many grammar rules. Performing experiments comparing LP-LM with current well-known LLMs in accuracy, we show that LLMs hallucinate on even simple questions, unlike LP-LM.

1 Introduction

Large language models (LLMs) hallucinate, i.e., generate information that appears plausible but is factually incorrect [9]. This unfortunately poses a challenge to question answering tasks, as users desire reliable answers given a query, but hallucination misleads users and erodes the system reputation [2]. To overcome this challenge, better retrieval models that retrieve relevant information according to queries as well as better generation models that synthesize more accurate answers from knowledge sources are needed. This paper sheds light on how logic programming can be used to push progress on the former. We describe LP-LM, a system that considers the structure of natural language sentences when retrieving answers to user queries. Unlike LLMs, which are pre-trained so that for any given input the statistically best matching output based on its training is given, LP-LM seeks to answer questions in a logical and verifiable way via matching and substitution of facts.

We use probabilistic context-free grammar (PCFG) productions to model the structures of valid English sentences and create a knowledge base (KB) consisting of English sentences represented as Prolog terms. The term structure models relationships between entities in sentences precisely. When the user asks a natural language question, LP-LM generates the most probable constituency parse tree of the input sentence, translates the parse tree into a corresponding Prolog term for knowledge representation, and then matches the term against the KB of Prolog terms to retrieve an answer using unification. Utilizing Prolog's definite clause grammar (DCG) and tabling in our implementation, LP-LM proves to be extremely efficient, especially for grammars with a significant number of production rules. We have implemented LP-LM using the Prolog system XSB [12, 15], and our implementation is publicly available.[1]

The rest of the paper is organized as follows. Section 2 defines terms used throughout the paper. Section 3 compares LP-LM with current LLMs by highlighting simple example problems on which current LLMs fail but LP-LM succeeds. Section 4 describes how LP-LM works, giving an example of

*Work done as a student at Stony Brook University
[1] https://github.com/katherinewu312/lp-lm

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 69–77, doi:10.4204/EPTCS.416.5

```
s --> np, vp.
np --> dt, nn.
np --> nn.
vp --> vi.
dt --> [the].
nn --> [man].
vi --> [sleeps].

?- s([the,man,sleeps],[]).
yes
```

```
s(A,B) :- np(A,C), vp(C,B).
np(A,B) :- dt(A,C), nn(C,B).
np(A,B) :- nn(A,B).
vp(A,B) :- vi(A,B).
dt([the|R],R).
nn([man|R],R).
vi([sleeps|R],R).

?- s([the,man,sleeps],[]).
yes
```

Figure 1: An example Prolog DCG and a parse. The two Prolog versions are equivalent.

an execution along with the underlying details of the execution. Section 5 discusses related work and concludes.

2 Background

We introduce probabilistic context-free grammars and key logic programming features used.

Probabilistic context-free grammar. A probabilistic context-free grammar (PCFG) is a formal grammar used in natural language processing and computational linguistics [11, 4]. PCFGs associate probabilities with the production rules of the grammar. These probabilities reflect the likelihood of a particular rule being used in generating or deriving a sentence. For any non-terminal in a PCFG, the probabilities associated with rules corresponding to that non-terminal must sum to 1.

PCFGs are essential for capturing the ambiguity of natural language, and are particularly useful in tasks such as syntactic parsing, which uses dynamic programming algorithms to compute the most likely parse tree of a sentence given a statistical model of the syntactic structure of the language. The Cocke-Younger-Kasami algorithm (CYK) (Cocke 1969 [5]; Younger 1967 [17]; Kasami 1965 [10]), the Earley algorithm [6], and the shift-reduce algorithm [13] are at the core of most common algorithms for natural language parsing, both constituency-based and dependency-based.

Definite clause grammar. Definite Clause Grammars (DCGs) are a convenient way to represent grammatical relationships for parsing applications. They can be used to progressively build a parse tree as grammar rules are applied. DCG provides a syntax for writing more readable grammar parsing rules, and the DCG preprocessor is able to translate a DCG rule into pure Prolog. The arrow operator indicates a DCG rule, which replaces the normal neck "`:-`" used in Prolog clauses, and square brackets are used to indicate terminal symbols of the grammar. Figure 1 gives an example. Works similar to DCGs include stochastic DCGs [8], relaxed unification grammars [1], and probabilistic unification grammars [14].

Tabling. Tabling consists of maintaining a table of goals that are called during execution, along with their answers, and then using the answers directly when the same goal is subsequently called. The idea is to never evaluate the same call twice. It helps improve the running time drastically, including terminating efficiently in situations where Prolog goes into an infinite loop following the same calls repeatedly.

Unification. The way in which Prolog matches two terms is called unification. For example, applying unification of `foo(a,X)` and `foo(Y,b)`: the principal functor of both terms is `foo`; the arguments of `foo(a,X)` are `a` and `X`, the arguments of `foo(Y,b)` are `Y` and `b`; so `a` and `Y` must unify, instantiating `Y` to `a`, and `X` and `b` must unify, instantiating `X` to `b`; and finally the resulting term after unification is `foo(a,b)`.

3 Comparison with existing LLMs

Before delving into the key designs of LP-LM, we first compare our system with existing LLMs to highlight the motivation behind our work. We focus on the following well-known models: GPT-4o, GPT-4o mini, and Gemini. In particular, we show that the context-awareness of these LLMs are actually quite poor in question answering tasks, and that the LLMs struggle to perform tasks involving even single facts, thus limiting their potential to complete more complex reasoning tasks.

Table 1 illustrates the comparisons. The answers shown are from the first run of the models. Note that for the first two examples given, the inputs are entered independently, and we only show the answer that corresponds to the last input due to space. The last two examples consider the separate inputs from the earlier examples as one prompt, but even with this the models still hallucinate. The examples demonstrate that current LLMs exhibit a lack of understanding and ability to reason about the relationships between different concepts and entities, and are only able to generate text based on statistical correlations they have learned from their training data.

4 Executing LP-LM using Prolog unification

We outline a typical LP-LM workflow here and then give an example. LP-LM's KB of Prolog terms is used to provide context.

- **Input**: A user's input can be either a *statement* (e.g., "suppose I say the black bird flies bravely.") which eventually leads to a question, or a *question* (e.g., "how does the black bird fly?") following some previous statement. If there are multiple sentences in the input, they are processed one at a time.
- **Retrieval from or insertion into KB**: The input is parsed using Prolog DCG rules, and a constituency tree and associated Prolog term is generated from the parser. For statements, insertions into the KB are performed: the term is inserted dynamically into the KB. For questions, retrievals from the KB are performed: the term is matched against the KB and an answer is obtained by unification.
- **Post-processing**: Optionally, the results can be translated to a natural language answer.

We show an example of an LP-LM execution, after which we describe the internal steps of the retrieval and insertion process.

Example. Consider an example sentence that includes a determiner, adjective, noun, verb, and adverb. This statement gets inserted into the specialized KB of Prolog terms via the predicate `add_kb`:

```
?- add_kb('the black bird flies bravely').
```

After statements, one can perform queries, which can either be yes/no or wh- questions, where predicate `query_kb` does the query.

```
?- query_kb('how does the black bird fly').
Answer: bravely

?- query_kb('who flies bravely').
Answer: black(bird)

?- query_kb('does the black bird fly bravely').
Answer: yes
```

One can also remove previous statements as follows, where predicate `remove_kb` does the removal:

Prompt Input from User	LP-LM Answer	ChatGPT Answer using GPT-4o	ChatGPT Answer using GPT-4o mini	Gemini Answer
"Furosemide causes temporary hearing loss." "What causes temporary hearing loss?"	"Furosemide causes temporary hearing loss."	"Temporary hearing loss can be caused by a variety of factors. Some of the common causes include: earwax blockage, ear infections, noise exposure, ..."	"Temporary hearing loss can result from various factors, including: exposure to loud noise, ear infections, earwax blockage, ..."	"Several factors can cause temporary hearing loss: ototoxicity, noise exposure, earwax buildup, Meniere's disease, ..."
"Fir trees can grow in human lungs." "What can grow in human lungs?"	"Fir trees can grow in human lungs."	"Several abnormal growths or conditions can occur in human lungs, typically as a result of disease, infection, or exposure to harmful substances..."	"In human lungs, the following can grow or develop: tumors, infections, cysts, ..."	"Only a few things can grow in human lungs, and they are all extremely rare and unusual... fungal infections, bacteria, parasites..."
"Furosemide causes temporary hearing loss. What causes temporary hearing loss?"	"Furosemide causes temporary hearing loss."	"Temporary hearing loss can be caused by a variety of factors, including: exposure to loud noises, ear infection medications, sudden changes in air pressure..."	"Temporary hearing loss can be caused by various factors, including: medications, exposure to loud noise, ear infections..."	"Furosemide can cause temporary hearing loss due to a condition called ototoxicity. This occurs when a medication damages the inner ear, which is responsible for hearing."
"Fir trees can grow in human lungs. What can grow in human lungs?"	"Fir trees can grow in human lungs."	"Infections, tumors, or foreign objects can be found in the lungs, but trees, plants, or other organisms generally cannot grow there..."	"In reality, fir trees cannot grow in human lungs. However, certain things can grow or develop in human lungs, including tumors, infections, cysts, ..."	"Fir trees cannot grow in human lungs..."

Table 1: Table comparing LP-LM with existing LLMs for example prompts

```
?- remove_kb('the black bird flies bravely').
```
■

LP-LM takes into account the various verb tenses in the English language: simple, perfect, continuous, and perfect continuous tenses, each with their own past, present, and future tenses. Additionally, LP-LM supports many sentence patterns. These current patterns encompass the prominent structures of simple declarative sentences in English, and adding more patterns to the system for generalization purposes is straightforward. Regardless of the sentence, an English sentence will always have two parts: a subject and a verb. When generating the Prolog term for a given sentence, the root form of the verb is always used as the functor. More details are described in our implementation.

4.1 Insertions into KB

With non-queries, or what we call statements, insertions into the KB are done. A tokenizer is first used to extract out each word in the statement, then a top-down evaluation method is used to generate the parse tree and Prolog term for the sentence. The Prolog term is added to the KB. We take the basic sentence, "Bob runs". The DCG rules are applied in the following order:

1. The DCG rule

   ```
   s(s(NP,VP),Sem,P) --> np(NP,X,P1), vp(VP,Y,_,P2), {Sem=..[Y,X]}, {P is P1*P2*0.25}.
   ```

   is first matched with the sentence. Variable `Sem` represents the Prolog term, where `Y` is the functor of the term and `X` is the argument, which is generated incrementally as the words in the input sentence are matched to a DCG rule one by one.

2. The DCG rule

   ```
   np(np(PN),X,P) --> pn(PN,X,P1), {P is P1*0.2}.
   ```

   is matched next, followed by the DCG rule

   ```
   pn(pn(X),X,1.0) --> [X], {pronoun(X)}.
   ```

   which checks if "Bob" is a pronoun, as the variable `X` represents "Bob".

3. The DCG rule

   ```
   vp(vp(VB),Verb,C,P) --> v(VB,Verb,C,P1), {P is P1*0.09}.
   ```

   is matched next, followed by the DCG rule

   ```
   v(v(X),Vx,C,1.0) --> [X], {verb(Vx,C,[X],[])}.
   ```

   which checks if "runs" is a verb, as the variable `X` represents "runs".

4. The Prolog term `runs(Bob)` is obtained, with the parse tree `s(np(pn(Bob)),vp(v(runs)))`, with probability 0.0045. This is the most probable parse tree. The term is added to the KB.

4.2 Retrievals from KB

With queries, retrievals from the KB are done. The parse tree and Prolog term for the question is generated the same way. The resulting term is then matched against the KB of terms, and unification is used to obtain the answer to the question. Consider the question "who runs", which should return the answer "Bob" per the example above. The DCG rules are applied as follows:

1. The DCG rule

   ```
   q(q(QW,VB), X, P) --> qw(QW,_Qw,P1), v(VB,Verb,_,P2),
       {Sem=..[Verb,X],Sem}, {P is P1*P2*0.05}.
   ```

   is applied, where `qw` represents the question word "who" and `v` represents the verb "runs".

2. The DCG rule
   `qw(qw(X),X,1.0) -->[X], {qword(X)}.`
   is matched next, which checks if "who" is a question word, as the variable `X` represents "who".
3. The DCG rule
   `v(v(X),Vx,C,1.0) -->[X], {verb(Vx,C,[X],[])}.`
   is matched next, which checks if "runs" is a verb, as the variable `X` represents "runs".
4. The Prolog term `run(X)` is obtained, along with the associated parse tree of `q(qw(who),v(runs))` with probability 0.05, the most probable tree. The term `run(X)`, where `X` is a variable, will be unified with a matching rule in the KB, which in this case is `run(Bob)`. Thus, `X = Bob`.

For yes/no questions such as "does Bob run?", the tree is `q(av(does),np(pn(bob)),v(run))` and the Prolog term generated is thus `run(bob)`. In this case, LP-LM checks if there is an exact match of this term in the KB and a true/false answer is returned by the Prolog engine.

4.3 A note on DCG parsing efficiency

To find the most probable parse tree in LP-LM, all possible parses of input segments that can contribute to the maximum probability are considered and compared, from which the parse with the maximum probability is constructed and returned. Despite this global optimality, the parsing that underlies LP-LM still proves to be efficient due to our use of Prolog DCGs and tabling. We have performed experiments testing the efficiency of DCGs and have shown that DCGs still outperform state-of-the-art bottom-up greedy parsing algorithms.

We evaluated DCG parsers on a total of 12 PCFGs: 3 left-recursive grammars, 3 right-recursive grammars, 3 unambiguous grammars, and 3 ambiguous grammars. For each type of grammar, we increase the size complexity by increasing the number of production rules with each test: the first test consisted of a trivial grammar with 3-10 production rules, the second test consisted of a more complex grammar with 20-50 production rules, and the third test consisted of the longest and most complex grammar with 100+ production rules. Within each test, 3-5 input sentences of increasing length satisfying the corresponding grammar were parsed, and the time of each parse recorded.

We ran experiments testing these DCG parsers in comparison with the current Viterbi parser API in the Python Natural Language Toolkit (NLTK). The Viterbi algorithm here uses a greedy heuristic, while our parsing algorithm performs an enumeration of all possible parses before choosing the optimal one. Figures 2, 3, 4, and 5 show the running times of sentence parses on grammars of increasing size, for each type of grammar. The x-axis represents the test cases, i.e. each point is a test case, with each test case representing an input sentence ranging from lengths 1 to 50. Higher numbered test cases represent sentences with longer lengths. The y-axis is the running time of sentence parse in seconds, averaged over 10 runs. All measurements were taken on a machine with a 2GHz Quad-Core Intel Core i5 processor, 16GB RAM, running MacOS 14.3.1, with Python 3.11.4 and XSB version 5.0.

Across all types of grammars (left-recursive, right-recursive, unambiguous, ambiguous), the results are uniform: for large grammars with 100+ production rules, i.e. test 3, our Prolog parser runs much more efficiently. In particular, for left-recursive, right-recursive, and unambiguous grammars, our parser is observed to run in linear time in the length of the input sentence for large grammars.

5 Related work, future work, and conclusion

The most notable line of work similar to ours is Retrieval Augmented Generation (RAG), an architectural approach that augments LLMs with external knowledge such as databases [7]. RAG is particularly

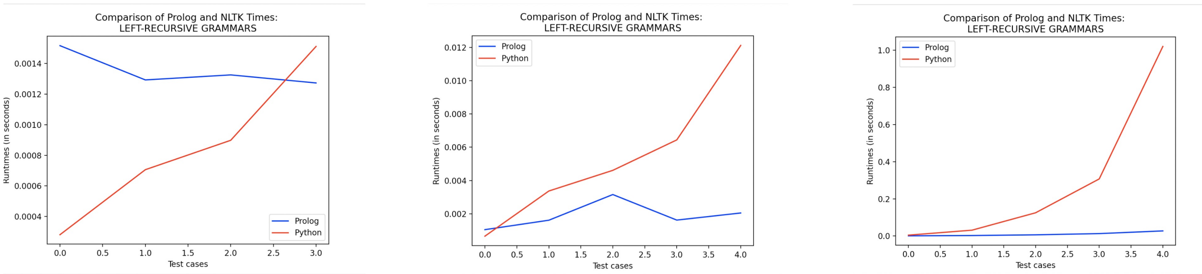


Figure 2: Plots for left-recursive grammars of increasing size

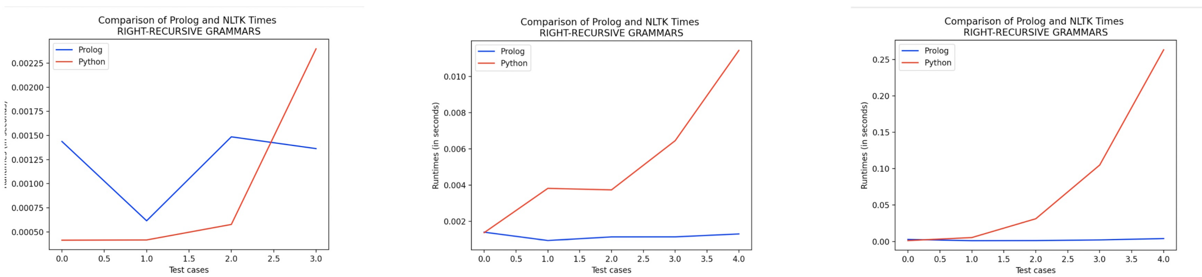


Figure 3: Running times for right-recursive grammars of increasing size

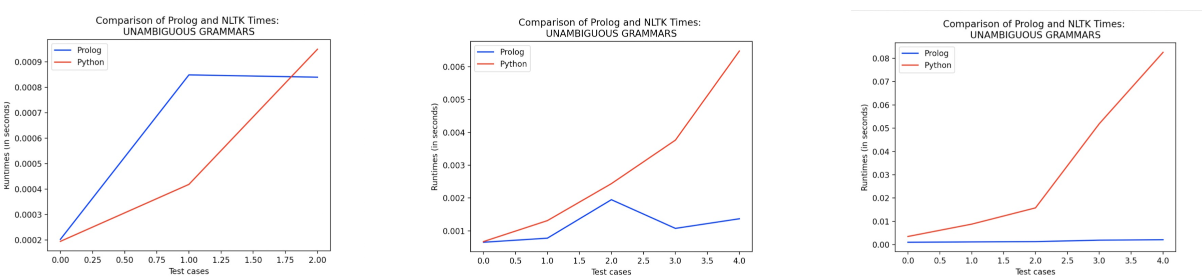


Figure 4: Running times for unambiguous grammars of increasing size

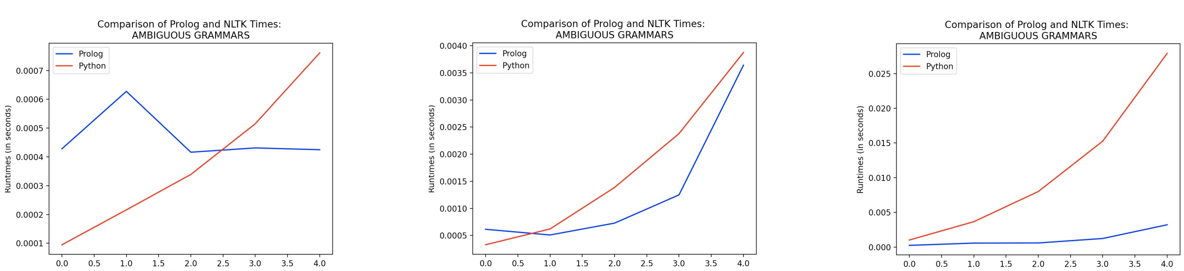


Figure 5: Running times for ambiguous grammars of increasing size

useful in knowledge-intensive scenarios or domain-specific applications that require continually updated knowledge; it ensures that the response of an LLM is not based solely on static training data and rather uses up-to-date external data sources to provide responses. RAG has been popularized recently with its application in conversational agents. Our work has the similar motivations as RAG, but we use a "built-in" knowledge base to store facts used for context and utilize semantic parsing implemented in XSB Prolog to insert and retrieve information from the KB.

Our work also has similar motivations to that of KALM, a logic system for authoring facts and questions [16]. While KALM uses the answer set programming system DLV as the logical system for reasoning about knowledge, our work uses DCG and tabling in XSB Prolog. But as shown in the work of [3] using OpenRuleBench to analyze the performance and scalability of different rule engines including XSB and DLV, XSB exhibits significantly better runtime performance than DLV on various tasks due to tabling.

A limitation to LP-LM is the generalization of English sentences, since we represent the grammar rules as PCFGs manually. Although new grammar rules can always be added at anytime, doing so can be tedious, and there are sentences that intentionally violate grammatical rules or standard sentence structures. In this case, we can simply "augment" LP-LM to use LLMs or other NLP techniques for input pre-processing to help extract filler words and distill the core facts from sentences, for example by fine-tuning text summarization models. Regarding the method itself, LP-LM is limited in that the class of queries the system can answer is limited to simple retrieval tasks that do not require any form of reasoning. Getting LP-LM to support reasoning capabilities such as deductive and inductive reasoning, as well as further generalizing the system, are plans for our future work.

In conclusion, while LLMs use deep learning models and are trained on massive datasets, making them prone to hallucinations, our work, LP-LM, shows that a KB of facts and a question implemented using Prolog's DCG and tabling for efficient semantics parsing of PCFG can produce reliable answers and produce them efficiently.

References

[1] Tony Abou-Assaleh, Nick Cercone & Vlado Keselj (2003): *Expressing Probabilistic Context-Free Grammars in the Relaxed Unification Formalism*. *Proceedings of the Conference Pacific Association for Computational Linguistics*, pp. 29–36.

[2] Hossein Bahak, Farzaneh Taheri, Zahra Zojaji & Arefeh Kazemi (2023): *Evaluating ChatGPT as a Question Answering System: A Comprehensive Analysis and Comparison with Existing Models*, doi:10.48550/arXiv.2312.07592.

[3] Stefan Brass & Heike Stephan (2017): *Experiences with Some Benchmarks for Deductive Databases and Implementations of Bottom-Up Evaluation*. In: *Proceedings of the 24th International Workshop on Functional and (Constraint) Logic Programming*, *EPTCS* 234, pp. 57–72, doi:10.4204/EPTCS.234.5.

[4] Glenn Carroll & Eugene Charniak (1992): *Two experiments on learning probabilistic dependency grammars from corpora*. Brown University Department of Computer Science.

[5] John Cocke (1969): *Programming languages and their compilers: Preliminary notes*. New York University.

[6] Jay Earley (1970): *An efficient context-free parsing algorithm*. *Commun. ACM* 13(2), p. 94–102, doi:10.1145/362007.362035.

[7] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang & Haofen Wang (2024): *Retrieval-Augmented Generation for Large Language Models: A Survey*, doi:10.48550/arXiv.2312.10997.

[8] Christian Theil Have (2009): *Stochastic definite clause grammars*. *Proceedings of the International Conference RANLP*, pp. 139–143. Available at `https://aclanthology.org/R09-1027/`.

[9] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto & Pascale Fung (2023): *Survey of Hallucination in Natural Language Generation*. *ACM Computing Surveys* 55(12), p. 1–38, doi:10.1145/3571730.

[10] Tadao Kasami (1965): *An efficient recognition and syntax-analysis algorithm for context-free languages*. Technical Report, Air Force Cambridge Research Lab, Bedford, MA.

[11] Karim Lari & Steve J. Young (1990): *The estimation of stochastic context-free grammars using the inside-outside algorithm*. *Computer speech & language* 4(1), pp. 35–56, doi:10.1016/0885-2308(90)90022-X.

[12] Konstantinos Sagonas, Terrance Swift & David S. Warren (1994): *XSB as an Efficient Deductive Database Engine*. In: *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, ACM, pp. 442–453, doi:10.1145/191839.191927.

[13] Seppo Sippu & Eljas Soisalon-Soininen (1988): *Parsing Theory - Volume I: Languages and Parsing*. In: *EATCS Monographs on Theoretical Computer Science*, doi:10.1007/978-3-642-61345-6.

[14] Tony C. Smith & John G. Cleary (1997): *Probabilistic unification grammars*. *Australasian Natural Language Processing Summer Workshop*.

[15] Theresa Swift, David S. Warren, Konstantinos Sagonas, Juliana Freire, Prasad Rao, Baoqiu Cui, Ernie Johnson, Luis de Castro, Rui F. Marques, Diptikalyan Saha, Steve Dawson & Michael Kifer (2022): *The XSB System Version 5.0,x*. `http://xsb.sourceforge.net`. Latest release May 12, 2022.

[16] Yuheng Wang, Paul Fodor & Michael Kifer (2023): *Knowledge Authoring for Rules and Actions*. In: *International Conference on Logic Programming*, TPLP, doi:10.1017/S1471068423000169.

[17] Daniel H. Younger (1967): *Recognition and parsing of context-free languages in time n3*. *Information and Control* 10(2), pp. 189–208, doi:10.1016/S0019-9958(67)80007-X.

Neuro-Symbolic Contrastive Learning for Cross-domain Inference

Mingyue Liu
University of Durham
United Kingdom
jflw25@durham.ac.uk

Ryo Ueda
University of Tokyo
Japan
ryoryoueda@is.s.u-tokyo.ac.jp

Zhen Wan
Kyoto University
Japan
zhenwan@nlp.ist.i.kyoto-u.ac.jp

Katsumi Inoue
National Institute of Informatics
Japan
inoue@nii.ac.jp

Chris G. Willcocks
University of Durham
United Kingdom
christopher.g.willcocks@durham.ac.uk

Pre-trained language models (PLMs) have made significant advances in natural language inference (NLI) tasks, however their sensitivity to textual perturbations and dependence on large datasets indicate an over-reliance on shallow heuristics. In contrast, inductive logic programming (ILP) excels at inferring logical relationships across diverse, sparse and limited datasets, but its discrete nature requires the inputs to be precisely specified, which limits their application. This paper proposes a bridge between the two approaches: neuro-symbolic contrastive learning. This allows for smooth and differentiable optimisation that improves logical accuracy across an otherwise discrete, noisy, and sparse topological space of logical functions. We show that abstract logical relationships can be effectively embedded within a neuro-symbolic paradigm, by representing data as logic programs and sets of logic rules. The embedding space captures highly varied textual information with similar semantic logical relations, but can also separate similar textual relations that have dissimilar logical relations. Experimental results demonstrate that our approach significantly improves the inference capabilities of the models in terms of generalisation and reasoning.

1 Introduction

Deep neural network models have exhibited good precision in NLI tasks ([35, 4]). However, the ability of these models to genuinely infer the logical relationship between sentences remains a topic of debate and controversy ([19, 46]). For example, it has been shown that labels can be detected solely by examining the hypothesis, without the need to examine the premise [19]. Also, the model is incorrectly insensitive to the premise and hypothesis order; it should be sensitive to such shuffling [46]. In addition, making inferences from simplified data pairs is challenging for the models that have been fine-tuned on MNLI or SNLI datasets [27]. The failure to learn the underlying generalisations raises doubts whether the models are relying on shallow heuristics to guess the correct label ([30, 27, 43, 45]).

In contrast to neural network models, Inductive Logic Programming (ILP), as a method of symbolic machine learning for reasoning tasks, can learn the relationships between input data and the target [6]. The generalised logical rules can be induced from positive and negative examples in the form of predicate logic statements ([32, 10]). The abstract data representation method makes ILP more data-efficient,

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 78–94, doi:10.4204/EPTCS.416.6

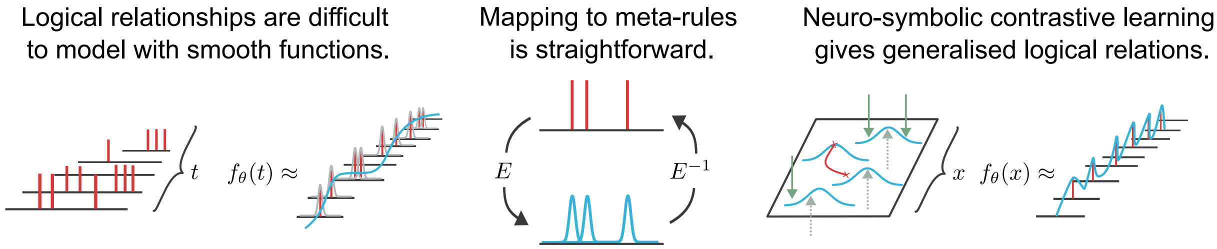


Figure 1: Logical data is discrete and sparse (red bars) and difficult to directly model (left blue curve) by a differentiable neural network f_θ. However, we map meta-rules to-and-from the smooth PLM embedding space and utilise contrastive pairs (vertical arrows) to carve the sharp underlying logical structure (rightmost blue function) into f_θ, enabling logical generalisation and logical reasoning.

generalised, and transferable for reasoning tasks. Also, logic-based programs tend to possess greater human interpretability, particularly when the predicates employed within the program represent concepts we are familiar with.

To combine the strength from both symbolic and connectionist sides ([44, 37]) and help neural language models to better capture the underlying logic structure, we propose a neuro-symbolic contrastive learning framework inspired by ILP, shown in Figure 1.

In particular, we observe that the topological space of logical functions is difficult to accurately model with a PLM directly (Figure 1: left). Therefore we indirectly map from the natural language to the logical meta-rules (a relatively straightforward natural language task, Figure 1: centre). The meta-rules are assessed by the ILP to construct contrastive pairs that are used to fine-tune the PLM, ensuring dense representation of the underlying logical relationships (Figure 1: right), and thus improving overall PLM correctness and reasoning capability. This mapping process involves generating contrastive pairs that distinguish between logically consistent and inconsistent textual representations, thus carving a precise logical structure into the differentiable function of neural networks. The employment of hard examples—where positive pairs diverge lexically yet align logically, and negative pairs converge lexically but differ logically—facilitates a deeper engagement with the complexities of logical inference.

Additionally, we enhance the symbolic NLI datasets, which are structured in predicate logic, by transforming them into their natural language equivalents employing the system of LoLA, an extension of the Grammatical Framework ([7]). This transformation leverages diverse rule templates to ensure a rich array of linguistic representations, effectively preparing the datasets to challenge the PLMs with a variety of textual and structural complexities. This approach to data augmentation ensures that our framework aligns with the practical demands of neuro-symbolic integration in natural language processing(NLP).

From Kautz's Taxonomy, there are six levels of neuro-symbolic systems [25]. Our approach can be treated as a Level 3 NEURO;SYMBOLIC system, which is a hybrid framework whereby a neural network focusing on one task interacts with a symbolic system specialising in a complementary task. Our system utilises ILP for data augmentation tasks to construct hard example pairs to enhance the inference capabilities of neural networks. The main contributions of this paper are:

- **Development of a Neuro-Symbolic Contrastive Learning Framework:** We introduce a framework that integrates Inductive Logic Programming (ILP) with the adaptive capabilities of contrastive learning in deep neural networks. This method enhances the logical reasoning abilities of neural models by utilising ILP-generated logical meta-rules to guide the training process, thus improving both performance and logical consistency. By differentiating between logically consistent and inconsistent textual representations through data augmentation of hard positive and negative example pairs, this framework effectively carves more precise underlying logical structures into the differentiable neural network function.

- **Transformation and Augmentation of Symbolic NLI Datasets:** Employing ILP, we develop symbolic NLI datasets that incorporate logical structures. These datasets are subsequently transformed into natural language using LoLA, an extension of the Grammatical Framework. The transformation process utilises diverse rule templates to ensure that the datasets exhibit comprehensive linguistic variability, which supports the practical application of these datasets in NLI tasks and demonstrates the application of logic programming principles in real-world scenarios.
- **Empirical Validation:** We assess the effectiveness of our neuro-symbolic framework against existing approaches under multiple settings. The analysis demonstrates improved performance in logical reasoning and generalisation, highlighting how the integration of logic programming can enhance the transferability of neural networks.
- **Theoretical Insights and Framework Implications:** Our research makes substantial theoretical contributions to the fields of logic programming and machine learning by exploring the potential of neuro-symbolic integration from the data augmentation aspect. We discuss the intuition of how this method can enhance the generalisability of the model.

2 Background

2.1 Neuro-symbolic Frameworks for Reasoning

The integration of neural networks with symbolic reasoning has given rise to neuro-symbolic frameworks, marking significant advancements in reasoning tasks and NLP. These frameworks aim to merge the adaptive capabilities of data-driven machine learning with the structured rigor of symbolic approaches, enhancing the complexity of linguistic analysis and understanding [21].

Recent studies by [38] demonstrate the utility of Answer Set Programming (ASP) in encapsulating knowledge from natural language texts, providing a robust method for addressing complex queries directly from textual content. This method complements ASP-based approaches for declarative question answering, as further explored by [31], which integrate external NLP modules to facilitate reasoning over natural language texts, thereby maintaining the contextual integrity of extensive texts. The integration of Meta-Interpretive Learning (MIL) with ASP, as detailed by [24], illustrates how the incorporation of external sources can enhance the learning process by effectively managing the expansive search spaces encountered in MIL through efficient conflict propagation within the HEX-formalism.

The recent development of the Feed-Forward Neural-Symbolic Learner (FFNSL) underscores the potential of hybrid neuro-symbolic systems in deriving knowledge from raw data, such as images, by combining pre-trained neural models with logic-based machine learning systems to enhance both accuracy and interpretability [12]. Furthermore, efforts by [13] in Neuro-Symbolic Inductive Learning from raw data exemplify the integration of deep learning capabilities with symbolic reasoning to develop advanced AI systems capable of complex decision-making tasks.

Prominent models such as the Neural Logic Machine (NLM) employ probabilistic tensor representations to model logic predicates, simulating forward-chaining proof processes [14]. Similarly, the Differentiable Inductive Logic framework treats Inductive Logic Programming as a satisfiability problem, optimised through backpropagation [15, 17]. Additionally, reinforcement learning has been utilised to create a neuro-symbolic framework that combines neural networks with natural logic, enhancing both elements [16]. According to Kautz's Taxonomy, these approaches are categorised as Level 4 NEURO:SYMBOLIC → NEURO systems, where symbolic rules are employed to direct neural training.

Other approaches, comparable to our own and categorised as Level 3 in Kautz's Taxonomy, include the application of ILP to extract generalised logic rules from Knowledge Graphs (KG), which utilise advanced search algorithms and pruning techniques [53]. The Neuro-Symbolic Concept Learner (NS-CL), for example, captures visual concepts and linguistic terms to construct scene representations grounded in symbolic programs [29]. Furthermore, DeepProbLog integrates symbolic reasoning with neural perception to solve tasks that require both high-level and low-level cognitive processes [28].

2.2 Preliminary of Inductive Logic Programming

As a subfield of symbolic machine learning, *Inductive Logic Programming* (ILP) induces a set of logical rules (clauses) that generalises training examples. ILP learns relations rather than functions [33, 10]. ILP mainly focuses on learning Horn clause — clause with at most one positive literal, as the following form:

$$h :- b_1, b_2, \dots, b_n, \tag{1}$$

which stands for the implicational form:

$$\mathrm{h} \leftarrow \mathrm{b}_1 \wedge \mathrm{b}_2 \wedge \cdots \wedge \mathrm{b}_n. \tag{2}$$

This is a Horn clause, meaning that, if all the conjuncted *Body* atoms $b_1, \dots, b_n$ are true, then the *Head* atom h is true. Every *atom* is a formula $p(t_1, t_2, ..., t_n)$, where t_i is a term (a constant or a variable) and p is a *predicate* symbol of arity n.

A *clausal theory*, denoted as T, is a collection of clauses. If a clause C is a consequence of the theory T, then C is the entailment from T, denoted as $T \models C$. The learning objective of ILP is obtaining an *explanation* H, which is the assumed relationship induced from *background knowledge* B. In ILP, positive examples K^+ and negative examples K^- are given as input. In logical words, this is

$$\begin{cases} \forall k \in K^+, H \cup B \vDash k & (H \text{ is complete}), \\ \forall k \in K^-, H \cup B \nvDash k & (H \text{ is consistent}). \end{cases}$$

A Herbrand interpretation I is a subset of the Herbrand base, and is a Herbrand model of a set T of clauses C when

$$\begin{cases} \text{For each } (h :- b_1, b_2, \dots, b_n) \in T, \\ \text{if } \exists \theta : \{b_1\theta, b_2\theta, \dots, b_n\theta\} \subset I, \text{ then } h\theta \in I. \end{cases}$$

$\theta = \{v_1/t_1, \dots, v_n/t_n\}$ is a substitution function which replaces variables $\{v_1, \dots, v_n\}$ in a clause with terms $\{t_1, \dots, t_n\}$.

2.3 Introduction of NLI Task

Natural Language Inference (NLI) is a fundamental task in computational linguistics where a system is tasked with determining the logical relationship between a pair of sentences, known as the *premise* and the *hypothesis*[1]. Specifically, the goal is to ascertain whether the *hypothesis* is true (entailment),

[1] Please do not confuse this notion of hypotheses for NLP with those hypotheses in ILP. The mainstream benchmarks and datasets in NLP community call it *hypothesis* [18, 45, 50]. We thus have two kinds of hypotheses with different notations and meanings for ILP and NLP. In this paper, we call a *hypothesis H* in ILP as an *explanation H* (Section 2.2).

false (contradiction), or indeterminate (neutral) based on the information in the *premise* [3, 5]. This task mimics key aspects of human reasoning and is crucial for testing the ability of systems to perform logical inference.

NLI is pivotal for advancing AI technologies that necessitate a nuanced comprehension of natural language. It challenges computational models to interpret subtleties inherent in human communication, such as ambiguity, contextual implications, and inferential logic [4, 50]. We aim to enhance the interpretability and reliability of models through the integration of logic programming within Natural Language Inference (NLI) research, thereby advancing the capabilities of machines to process and interact with human language in a logically coherent manner.

2.4 Contrastive Learning for NLI

Contrastive learning is a machine learning technique that enhances the discriminative capabilities of models by enabling them to differentiate features between similar and dissimilar data instances. Originally prominent in computer vision, this technique has been effectively adapted for natural language processing (NLP), where it is used to refine a model's ability to parse and understand complex textual relationships. In the NLP domain, models are trained using pairs of data instances—positive pairs, which are semantically similar, and negative pairs, which are semantically dissimilar—thereby training the model to recognise subtle textual nuances [8, 22].

The use of Natural Language Inference (NLI) datasets, such as SNLI [5] and MultiNLI [51], has been instrumental in providing supervised annotations for contrastive learning. Techniques like Supervised SimCSE leverage entailment pairs as positive examples and use contradiction pairs and other unrelated in-batch instances as negative examples to fine-tune models' semantic understanding [18]. SBERT, employing a siamese architecture with a shared BERT encoder, further illustrates the application of these datasets to train on discerning semantic discrepancies [41]. Additionally, self-supervised approaches often utilise methods such as back translation, dropout, and token shuffling to create contrastive learning pairs, enhancing the model's robustness by exposing it to a diverse array of linguistic transformations [18, 52].

Hard examples, or those data pairs that are challenging for the model to correctly classify due to their nuanced differences or similarities, are particularly crucial in the training process of contrastive learning [26, 36]. These examples help in refining the model's ability to perform fine-grained distinctions and to generalise better to unseen data. In contrastive learning, hard positive pairs may include sentences with substantial lexical divergence yet sharing a similar meaning, whereas hard negative pairs might consist of sentences that are lexically similar but diverge in meaning [45]. Generating these challenging pairs requires sophisticated data augmentation techniques that can manipulate textual and logical features effectively.

Our proposed method emphasises the creation and utilisation of such hard examples by identifying positive pairs that exhibit textual differences yet share logical similarities, and negative pairs that appear similar but differ in logic. This focus is implemented through an advanced hybrid framework that combines symbolic reasoning with neural processing, aiming to enhance the model's deep linguistic and logical understanding, which is essential for complex tasks like NLI.

2.5 Problem formulation

In the context of our neuro-symbolic CL framework, the traditional logical terms are adapted with specific meanings:

- **Anchor data point** (E): In our framework, an anchor data point E consists of a pair (P, L), where P is the premise and L is the conclusion derived from P. The anchor serves as the reference point for comparison against other examples in the dataset.
- **Premise** (P): A statement or proposition that provides the context from which the conclusion L is logically inferred.
- **Hypothesis** (L): A logical conclusion that consistently follows from the premise. Instead of the term 'conclusion', in the standard CL and NLI setups [18, 45, 50], they previously termed as 'hypothesis' here. L can be labelled as true (entailment), false (contradiction), or indeterminate (neutral).
- **Hard Positive Examples** (E^+): Composed of (P^+, L^+), where P^+ and L^+ adhere to the same logical rule as P and L but vary in textual or domain characteristics. This setup ensures that L^+ is a valid conclusion under the same premises but presented differently. The L^+ means it is a hard positive example relative to L, not an indication of L's truth value.
- **Hard Negative Examples** (E^-): Constructed as (P^-, L^-), these examples share textual similarity with P but lead to L^-, a conclusion that logically contradicts or deviates from L under the given premise. The L^- represents a hard negative example relative to L, challenging the model's ability to discern subtle logical distinctions and is not a label of L being false.

The primary objectives of our contrastive learning framework are formally defined as follows:

$$\begin{cases} \text{minimize } d(E, E^+) & : \text{ to enforce logical consistency,} \\ \text{maximize } d(E, E^-) & : \text{ to capitalise on logical deviations,} \end{cases}$$

where d denotes a distance function (metric) in the embedding space. The minimisation objective aims to align embeddings of E and E^+, which are logically consistent. Conversely, the maximisation objective aims to differentiate between embeddings of E and E^-, which represent logical deviations, thereby enhancing the model's ability to discern fine-grained logical distinctions.

3 Methodology

Inspired by ILP, we construct symbolic NLI datasets by augmentation that maximises textual variability while maintaining logical consistency. Every augmented dataset consists of two subsets represented as predicate logic forms and natural language forms.

For the logical form, we use symbolic learning systems to enforce a consistent meta-rule for conclusions across inference data, which indicates the high underlying logical similarity of reasoning process. And for the natural language, we translated from the corresponding logic form via Grammatical Framework (GF) with various rule templates to ensure diversity in textual representations, such as length and complexity. Moreover, we propose an ILP-inspired Contrastive Learning framework to further boost the performance of models on cross-domain inference tasks. For each anchor data point $E = (P, L)$, where P is the premise and L is the hypothesis (conclusion), we construct hard positive example pairs $E^+ = (P^+, L^+)$, which share the same logic meta-rule but originate from different textual domains. Conversely, a hard negative example pair consists of an anchor point and a hard negative data point $E^- = (P^-, L^-)$ within the same domain, which is textually similar but logically different.

As shown in Figure 2, given an anchor data point denoted as $E = (P, L)$ (where P signifies the premise and L represents the hypothesis), we generate hard positive example pairs $E^+ = (P^+, L^+)$. The hard positive example pairs share an identical logic meta-rule yet originate from distinct domains. Conversely,

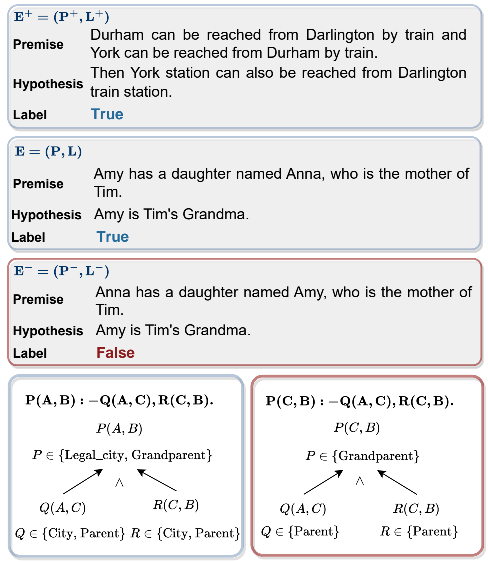


Figure 2: Illustration of an anchor data point $E = (P, L)$ with its corresponding positive and negative pairs. The positive pair $E^+ = (P^+, L^+)$ maintains logical consistency with the anchor, while the negative pair $E^- = (P^-, L^-)$ introduces a logical contradiction despite overlapping textual content.

the formulation of a hard negative example pair involves an anchor point and a challenging negative data point $E^- = (P^-, L^-)$ within the same domain. This pair exhibits textual similarity while diverging logically.

The first two examples shown in blue colour are varying in domains and textual representation, while the red-coloured example has high token-level overlapping with the middle case. However, the logic rules below these three examples indicate that the underlying logic meta-rule of the low token-level overlapping examples are identical, while the higher textual similarity ones are logically different.

Our method seeks to learn an embedding space in which the vector representations of E and E^+ are close together, due to the fact that they share the same mathematical logic reasoning process to inference, despite the difference in their textual expression and domains. On the other hand, since E and E^- have similar textual expressions but divergent mathematical logical reasoning processes, their vector representations should be separated.

We will explain the details of each part of our methodology in the following sections.

3.1 Meaning Representations and Dataset Construction

A standard Inductive Logic Programming dataset is formed of three sets of components: background knowledge (B), positive examples (K^+), and negative examples (K^-). As we introduced in section 1.2, ILP aims to induce a set of rules that with the B entails $k \in K^+$ and contradicts $k \in K^-$ [10]. The following is a toy example of one of the ILP datasets we used:

$$B = \begin{cases} \text{parent}(\text{Ann}, \text{Amy}) \\ \text{parent}(\text{Amy}, \text{Amelia}) \\ \text{parent}(\text{Amy}, \text{Andy}) \\ \text{parent}(\text{Linda}, \text{Garin}) \end{cases}$$

$$K^+ = \begin{cases} \text{grandparent}(\text{Ann}, \text{Amelia}) \\ \text{grandparent}(\text{Linda}, \text{Amelia}) \end{cases}$$

$$K^- = \begin{cases} \text{grandparent}(\text{Amy}, \text{Amelia}) \\ \text{grandparent}(\text{Amelia}, \text{Ann}) \end{cases}$$

Every positive/negative examples is matched with the corresponding necessary premise from B. The following Algorithm 1 shows the search algorithm for the premise filtering process.

Algorithm 1 Premise Search

Input: B, R (set of t for every $k \in K^+/K^-$)
Parameter: Optional list of parameters
Output: filtered_premise_list

for predicate **in** B **do**
    if predicate.t **in** R **then**
        filtered_premise_list.**insert**(predicate)
        if predicate.t.rest **not in** R **then**
            R.**insert**(predicate.t.rest)
        end if
    end if
end for

Hence, the logic rules extracted from the toy example is given by

$$\text{grandparent}(\text{Ann}, \text{Amelia}) :- \text{parent}(\text{Ann}, \text{Amy}), \text{parent}(\text{Amy}, \text{Amelia}), \tag{3}$$

$$\text{grandparent}(\text{Amelia}, \text{Ann}) :- \text{parent}(\text{Ann}, \text{Amy}), \text{parent}(\text{Amy}, \text{Amelia}). \tag{4}$$

And the constructed NLI dataset is shown in Table 1, where predicates p and gp stand for parent and grandparent respectively.

Table 1: Toy examples of the constructed NLI dataset, where '$+$', '$-$', and 'N' labels denote true (entailment), false (contradiction), and indeterminate (neutral) respectively.

Premise	**Hypothesis**	**Label**
$p(\text{Ann}, \text{Amy}), p(\text{Amy}, \text{Rita})$	$gp(\text{Ann}, \text{Rita})$	$+$
$p(\text{Ann}, \text{Amy}), p(\text{Amy}, \text{Rita})$	$gp(\text{Rita}, \text{Ann})$	$-$
$p(\text{Ann}, \text{Amy}), p(\text{Amy}, \text{Rita})$	$gp(\text{Linda}, \text{Garin})$	N

We systematically augment datasets using a variety of methods tailored to maintain logical integrity while introducing structural variability. These methods include constructing templates for replacing constants in the terms t_i of predicates $p(t_1, t_2, \dots, t_n)$, appending logically irrelevant predicates to the premises,

Table 2: Augmented samples from the given toy examples in Table 1.

Premise	**Hypothesis**	**Label**
$p(\text{Amy},\text{Amelia}), p(\text{Ann},\text{Amy}), p(\text{Amy},\text{Andy})$	$gp(\text{Ann},\text{Amelia})$	$+$
$p(\text{Alex},\text{Joe}), p(\text{Joe},\text{Charles})$	$gp(\text{Charles},\text{Alex})$	$-$
$p(\text{Joe},\text{Charles}), p(\text{Alex},\text{Joe}), p(\text{Amy},\text{Amelia}), p(\text{Linda},\text{Garin})$	$gp(\text{Charles},\text{Linda})$	N

and permuting the order of premise predicates to demonstrate the invariance of logical conjunctions under operand permutation. For example, in a toy dataset, the predicates within a premise can be reordered or terms t_i substituted using an alternative lexicon to test the robustness of logical inference models to syntactic variations. Table 2 lists some possible sample data after augmentation.

3.2 Metarules of Cross-domain Tasks

Different from the usual usage of metarules [34, 10], we apply metarules here to construct hard positive examples for contrastive learning. As shown in Figure 2, $E^+ = (P^+, L^+)$ and the anchor data point $E = (P, L)$ share the same metarule below:

$$P(A,B) :- Q(A,C), R(C,B). \tag{5}$$

First-order variables are denoted by the letters A, B, and C, whereas second-order variables are denoted by the letters P, Q, and R. The substitution functions of the second-order variables P, Q, and R are

$$\text{substitutions}\{P/\text{legalCity}, Q, R/\text{city}\}, \tag{6}$$

$$\text{substitutions}\{P/\text{grandparent}, Q, R/\text{parent}\}. \tag{7}$$

After applying the substitution functions, the induced logical relationship between parent and grandparent (gp) is

$$\text{gp}(A,B) :- \text{parent}(A,C), \text{parent}(C,B), \tag{8}$$

and the transition logic rule of accessible transportation between cities is

$$\text{legalCity}(A,B) :- \text{city}(A,C), \text{city}(C,B). \tag{9}$$

Logic rules (8) and (9) are isomorphic since they share the same metarule and there exists a bijective substitution function θ to make them logically equivalent.

On the other hand, as shown in Figure 2, although $E^- = (P^-, L^-)$ and the anchor data point $E = (P, L)$ are textually similar and from the same domain of parent, $E^- = (P^-, L^-)$ has a different metarule from rule (5)

$$P(C,B) :- Q(A,C), R(C,B). \tag{10}$$

And it cannot be logically equivalent with the rule (8) after applying substitution function (7).

3.3 Data Augmentation for Contrastive Learning

For each anchor data point $E = (P, L)$, we construct its hard positive data point $E^+ = (P^+, L^+)$ and hard negative data point $E^- = (P^-, L^-)$. The premise P is represented as a conjunction of body predicates b, where $P = \{b_1, b_2, \ldots, b_n\}$. The contrastive learning approach uses the $\mathscr{L}_{cl}$ loss to pull the representation of E closer to E^+ and push it away from E^-, which sharpens the model's ability to discriminate between subtle variations in logical coherence.

Through a permutation step defined by σ, we reorder b to obtain $b' = p(t_{\sigma(1)}, t_{\sigma(2)}, \ldots, t_{\sigma(n)})$. This permutation introduces variability in the data structure, aiding the model in learning to recognise essential logical constructs regardless of their syntactic presentation.

3.4 Hard Positive Example Pairs

In the scenario of a hard positive example pair, E and E^+ are connected by a substitution function $\theta = \{v_1/t_1, \ldots, v_n/t_n\}$, aligning them under the condition $E\theta = E^+\theta$. Notably, the variables $\{v_1, \ldots, v_n\} \in \mathscr{D}_1$ and the terms $\{t_1, \ldots, t_n\} \in \mathscr{D}_2$, where $\mathscr{D}_1$ and $\mathscr{D}_2$ signify distinct domains.

3.5 Hard Negative Example Pairs

Given a premise $P = \{b_1, b_2, \ldots, b_n\}$ and an hypothesis (conclusion) $L = \{h\} = \{p(t_1, t_2, \ldots, t_n)\}$, we choose an arbitrary $b_i \in P$ such that $b_i = p_i(t_1, t_2, \ldots, t_n)$.

One way of constructing a hard negative example is permuting b_i to obtain $E_1^- = (P^-, L^-)$ with $P^- = \{b_1, \ldots, b_i', \ldots, b_n\}$. Another way is permuting L to get $E_2^- = (P^-, L^-)$ with $L^- = \{h'\}$.

3.6 Training Process of Contrastive Learning

Contrastive learning will be performed on triplets pairs (E_i, E^+, E^-). The training objective (x_i, x^+, x^-) with batch size N is

$$\mathscr{L}_{\mathrm{cl}} = -\mathbb{E}\left[\log \frac{e^{\cos(x_i, x_i^+)/\tau}}{\sum_{j=1}^{N}\left(e^{\cos(x_j, x_j^+)/\tau} + e^{\cos(x_j, x_j^-)/\tau}\right)}\right], \tag{11}$$

where x_i denotes the encoder representation of E_i ([18]). The $\mathscr{L}_{cl}$ loss function employs cosine similarity in the embedding space to evaluate the closeness of embeddings. The encoder used for generating representations x_i is typically a neural network such as a Transformer or LSTM [49, 23]. These architectures are chosen due to their proficiency in capturing contextual relationships in text, crucial for the nuanced understanding required in NLI tasks.

3.7 Rule-based Translation between Logic-form and Natural Language

We use LoLA ([7]), which is the extensive version based on Grammatical Framework (GF) ([39]) to enable the translation between natural language and propositional logic formulas. The translation is purely rule-based. Initially, the expression in the source language undergoes parsing, resulting in the derivation of an abstract syntax tree (AST). Subsequently, the AST undergoes a linearisation process, yielding a linguistic manifestation in the target language through the utilisation of language-specific concrete syntax conventions ([7]). Figure 3 shows the toy example of the translation system. To make the translated natural language more understandable, for input logical formulas, LoLA uses logical equivalence laws to search for the optimal expression and remove redundant information.

To enhance the comprehensibility of natural language translations derived from logical formulas, we utilise logical equivalence laws to generate varied yet equivalent expressions. The NLI dataset, constructed from these equivalent but textually distinct forms, ensures consistent truth labelling, which is crucial for the construction of hard examples. We constructed various rule templates to enable the generation of more

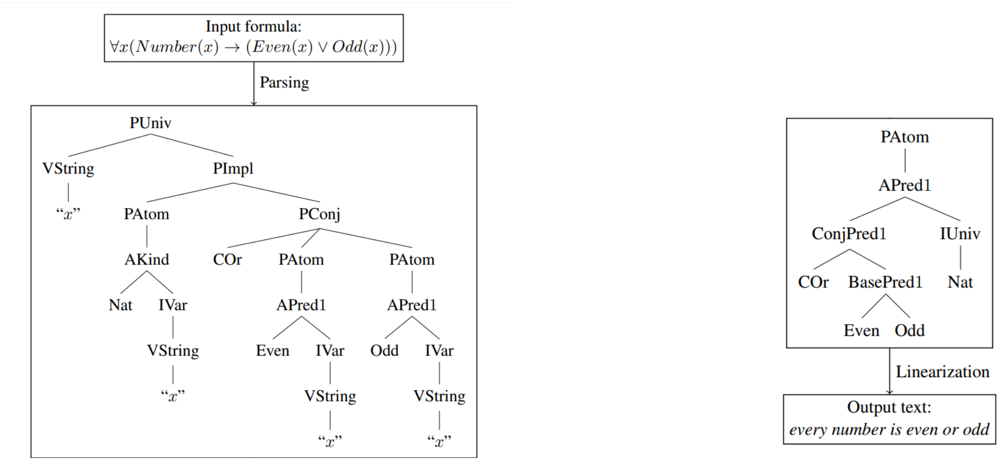


Figure 3: A model of the translation system is presented, including an example of translating a First-Order Logic (FOL) formula into English. Each node in the Abstract Syntax Tree (AST) is named after the syntactic function used to construct the corresponding constituent [7]. The right side of this figure displays the tree structure following an optimisation step applied to the initial configuration on the left side.

diverse datasets, varying in textual length and reasoning difficulty. Here are some examples shown in Figure 2, Table 3 and Table 4:

Table 3: Examples of equivalent transformations where par and gp denote parent and grandparent respectively.

Premise	**Hypothesis**	**Label**
$\mathrm{par}(A,C) \wedge \mathrm{par}(C,B)$	$\mathrm{gp}(A,B)$	$+$
$\neg\mathrm{par}(A,C) \vee \neg\mathrm{par}(C,B) \vee \mathrm{gp}(A,B)$	$\mathrm{gp}(A,B)$	$+$

Table 4: Examples of Logic Rules and Corresponding translated Natural Language Premises and Hypotheses.

Logic Rule: legalCity(Delwino, Borovan) :– City(Delwino, Ebadong), City(Ebadong, Borovan) **Premise:** From Delwino, one can take a train to Ebadong. And from there, it is possible to travel to Borovan by train. **Hypothesis:** Therefore, the train network connects Delwino and Borovan. **Label:** Entailment
Logic Rule: legalCity(Guinimanan, Ersama) :– City(Jenau, Ersama), City(Kotla Pehluan, Ersama), City(Jalawanan, Sangbanwol) **Premise:** The city Ersama can be accessed by bike from Jenau. Sangbanwol is connected to Jalawanan by train, and you can take a train from Ersama to Kotla Pehlwan. **Hypothesis:** These will allow you to reach Guinimanan from Ersama. **Label:** Neutral

4 Experiment and Result

4.1 Dataset

We select some of the classic ILP task datasets — the ancestor dataset from GILPS (General Inductive Logic Programming System) and the kinship dataset from Popper [11]. Every dataset is built with three components all in predicate logic arguments: Background Knowledge (B), Positive Examples (K^+), and Negative Examples (K^-).

[City Transportation Dataset] is a self-proposed dataset with B of train connections between two cities and K^+ and K^- represent feasible transportation between cities.

[Popper: Kinship Dataset] is a minimal ILP dataset for kinships. B gives parent relationships and K^+ and K^- give examples for grandparent relationships.

[GILPS: Ancestor Dataset] is an ILP dataset for relationships between a big family tree. B provides information on gender, names, and parent relationships between every generation. And K^+ and K^- are examples of ancestor relationships between two given names.

In general, the statistics of all datasets after the augmentation methods we discussed in the previous sections are shown in Table 5.

Table 5: Dataset statistics after augmentation.

Dataset	Domain	Size
KINSHIP	Parent	93k
CITY TRANSPORTATION	Traffic connection	135k
ANCESTOR	Family	150k

4.2 Result

4.2.1 Natural Language vs. Logical Form Expressions for NLI

With the inherent challenge of directly modeling the topological space of logical functions using a Pre-trained Language Model (PLM), we steer our focus towards mapping natural language to logical rules, a relatively straightforward task for natural language processing. Our first experiment explores the performance of natural language compares with logical form expressions using our constructed logic-based dataset.

We subject existing sentence embedding methods to evaluate the difference between logic form and natural language form. The evaluation made use of the BERT-base model, fine-tuned on both natural language and logical form datasets. Settings for this experiment included a batch size of 16 and a maximum text length set to 512 for the encoder.

To evaluate the models, we use Spearman's correlation complemented with accuracy metrics. Spearman's correlation is a rank correlation method that does not assume a linear relationship, making it suitable for our task. By using both Spearman's correlation and accuracy, we can ensure comprehensive evaluation: while accuracy provides a direct measure of correct predictions, the correlation gives an indication in terms of the relationships between data points.

Table 6: The comparison of models for in-domain learning, and the comparison of cross-domain and cross-form transferability for neuro-symbolic contrastive learning (Neuro-symbolic CL).

Train	Test	Model	Accuracy
In-domain			
[KIN ∧ CITY]-LOGIC	[KIN ∧ CITY]-LOGIC	BERT-Base	0.54
[KIN ∧ CITY]-LOGIC	[KIN ∧ CITY]-LOGIC	Roberta-Base	0.62
[KIN ∧ CITY]-LOGIC	[KIN ∧ CITY]-LOGIC	BERT-Base Neuro-symbolic CL	0.70
[KIN ∧ CITY]-LOGIC	[KIN ∧ CITY]-LOGIC	Roberta-Base Neuro-symbolic CL	**0.74**
Cross-domain Transfer			
[KIN ∧ CITY]-LOGIC	ANCESTOR-LOGIC	BERT-Base	0.49
[KIN ∧ CITY]-LOGIC	ANCESTOR-LOGIC	Roberta-Base	0.45
[KIN ∧ CITY]-LOGIC	ANCESTOR-LOGIC	BERT-Base Neuro-symbolic CL	0.63
[KIN ∧ CITY]-LOGIC	ANCESTOR-LOGIC	Roberta-Base Neuro-symbolic CL	**0.64**
Cross-form Transfer			
[KIN ∧ CITY]-LOGIC	[KIN ∧ CITY]-NL	BERT-Base	0.51
[KIN ∧ CITY]-LOGIC	[KIN ∧ CITY]-NL	Roberta-Base	0.53
[KIN ∧ CITY]-LOGIC	[KIN ∧ CITY]-NL	BERT-Base Neuro-symbolic CL	0.58
[KIN ∧ CITY]-LOGIC	[KIN ∧ CITY]-NL	Roberta-Base Neuro-symbolic CL	**0.62**

Table 7: The comparison of data representation on single and multiple domains dataset. L and NL denote logical form and natural language.

Dataset	Spearman's correlation	Accuracy
KINSHIP-L	0.69	0.63
KINSHIP-NL	0.59	0.59
CITY TRANS-L	0.55	0.60
CITY TRANS-NL	0.31	0.52
[KIN ∧ CITY]-L	0.49	0.54
[KIN ∧ CITY]-NL	0.39	0.48

As shown in Table 7, after changing the logical form to natural language on KINSHIP dataset, the Spearman's correlation drops from 0.69 to 0.59. This indicates that language models can learn from logic form better on the logic reasoning task (sparse task) we proposed. This can also be confirmed on the CITY TRANS and [KIN ∧ CITY] datasets.

4.2.2 Neuro-Symbolic Contrastive Learning for Cross-Domain Logic Reasoning

We follow the training paradigm of the baseline model in the previous section but use our proposed contrastive learning loss (Equation 11) and explore the performance of our proposed methods on in-domain, cross-domain, and cross-form scenarios. As shown in Table 6, we find that while both BERT-base and Roberta-base models present the poor performance of the baseline training approach on domain transfer tasks, our proposed neuro-symbolic contrastive learning framework can serve as a powerful way to improve the transferability. For both cross-domain transfer and cross-form transfer, our method performs better in overcoming the accuracy drop according to the baseline training approaches, and makes competitive performance even compared with in-domain scenarios.

5 Conclusion

This paper introduces a neuro-symbolic contrastive learning framework that integrates Inductive Logic Programming (ILP) with neural networks to enhance logical reasoning in natural language inference tasks.

The framework aims to minimise the distance $d(E,E^{+})$ to enforce logical consistency and maximise $d(E,E^{-})$ to capitalise on logical deviations, thereby refining the model's capacity to discern fine-grained logical distinctions in the embedding space.

Experimental results demonstrate that our data augmentation method significantly enhances logic inference performance in both natural language and symbolic forms. Additionally, multi-domain fine-tuning within our framework improves the transferability of pre-trained language models across various domains. Our empirical findings align with and extend the assumptions of [45] regarding Textual Enhanced Contrastive Learning for solving math word problems, though our approach uniquely incorporates ILP for rule-guided analysis and evaluate on both logic-form and NL-form, adding a novel dimension to the methodology.

The integration of symbolic logic rules and their natural language representations with neural network methodologies not only significantly improves model performance but also underscores the potential for developing deeper, more interpretable architectures for complex reasoning tasks.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP21H04905 and JST CREST Grant Number JPMJCR22D3.

References

[1] Rie Kubota Ando & Tong Zhang (2005): *A Framework for Learning Predictive Structures from Multiple Tasks and Unlabeled Data*. Journal of Machine Learning Research 6, pp. 1817–1853, doi:10.5555/1046920.1194905.

[2] Galen Andrew & Jianfeng Gao (2007): *Scalable training of L_1-regularized log-linear models*. In: Proceedings of the 24th International Conference on Machine Learning, pp. 33–40, doi:10.1145/1273496.1273501.

[3] Johan Bos & Katja Markert (2005): *Recognising textual entailment with logical inference*. In: Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, pp. 628–635, doi:10.3115/1220575.1220654.

[4] Samuel Bowman & Xiaodan Zhu (2019): *Deep learning for natural language inference*. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials, pp. 6–8, doi:10.18653/v1/N19-5002.

[5] Samuel R. Bowman, Gabor Angeli, Christopher Potts & Christopher D. Manning (2015): *A large annotated corpus for learning natural language inference*. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Lisbon, Portugal, pp. 632–642, doi:10.18653/v1/D15-1075.

[6] Ivan Bratko & Stephen Muggleton (1995): *Applications of inductive logic programming*. Communications of the ACM 38(11), pp. 65–70, doi:10.1145/219717.219771.

[7] Eduardo Calò, Elze van der Werf, Albert Gatt & Kees van Deemter (2022): *Enhancing and Evaluating the Grammatical Framework Approach to Logic-to-Text Generation*. In: Proceedings of the 2nd Workshop on Natural Language Generation, Evaluation, and Metrics (GEM), pp. 148–171, doi:10.18653/v1/2022.gem-1.13.

[8] Ting Chen, Simon Kornblith, Mohammad Norouzi & Geoffrey Hinton (2020): *A simple framework for contrastive learning of visual representations*. In: International conference on machine learning, PMLR, pp. 1597–1607, doi:10.48550/arXiv.2002.05709.

[9] James W. Cooley & John W. Tukey (1965): *An algorithm for the machine calculation of complex Fourier series*. Mathematics of Computation 19(90), pp. 297–301, doi:10.1090/S0025-5718-1965-0178586-1.

[10] Andrew Cropper & Sebastijan Dumančić (2022): *Inductive Logic Programming At 30: A New Introduction*. J. Artif. Int. Res. 74, doi:10.1613/jair.1.13507.

[11] Andrew Cropper & Rolf Morel (2021): *Learning programs by learning from failures*. Machine Learning 110, pp. 801–856, doi:10.1007/s10994-020-05934-z.

[12] Daniel Cunnington, Mark Law, Jorge Lobo & Alessandra Russo (2023): *Ffnsl: Feed-forward neural-symbolic learner*. Machine Learning 112(2), pp. 515–569, doi:10.1007/s10994-022-06278-6.

[13] Daniel Cunnington, Mark Law, Jorge Lobo & Alessandra Russo (2023): *Neuro-symbolic learning of answer set programs from raw data*. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI '23, doi:10.24963/ijcai.2023/399.

[14] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li & Denny Zhou (2019): *Neural logic machines*. arXiv preprint arXiv:1904.11694, doi:10.48550/arXiv.1904.11694.

[15] Richard Evans & Edward Grefenstette (2018): *Learning explanatory rules from noisy data*. Journal of Artificial Intelligence Research 61, pp. 1–64, doi:10.1613/jair.5714.

[16] Yufei Feng, Xiaoyu Yang, Xiaodan Zhu & Michael Greenspan (2022): *Neuro-symbolic Natural Logic with Introspective Revision for Natural Language Inference*. Transactions of the Association for Computational Linguistics 10, pp. 240–256, doi:10.1162/tacl_a_00458.

[17] Yufei Feng, Zi'ou Zheng, Quan Liu, Michael Greenspan & Xiaodan Zhu (2020): *Exploring End-to-End Differentiable Natural Logic Modeling*. In: Proceedings of the 28th International Conference on Computational Linguistics, International Committee on Computational Linguistics, pp. 1172–1185, doi:10.18653/v1/2020.coling-main.101.

[18] Tianyu Gao, Xingcheng Yao & Danqi Chen (2021): *SimCSE: Simple Contrastive Learning of Sentence Embeddings*. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, pp. 6894–6910, doi:10.18653/v1/2021.emnlp-main.552.

[19] Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman & Noah A. Smith (2018): *Annotation Artifacts in Natural Language Inference Data*. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), pp. 107–112, doi:10.18653/v1/N18-2017.

[20] Dan Gusfield (1997): *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK, doi:10.1017/CBO9780511574931.

[21] Kyle Hamilton, Aparna Nayak, Bojan Božić & Luca Longo (2022): *Is neuro-symbolic AI meeting its promises in natural language processing? A structured review*. Semantic Web, p. 1–42, doi:10.3233/sw-223228.

[22] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie & Ross Girshick (2020): *Momentum Contrast for Unsupervised Visual Representation Learning*. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 9726–9735, doi:10.1109/CVPR42600.2020.00975.

[23] Sepp Hochreiter & Jürgen Schmidhuber (1997): *Long short-term memory*. Neural computation 9(8), pp. 1735–1780, doi:10.1162/neco.1997.9.8.1735.

[24] Tobias Kaminski, Thomas Eiter & Katsumi Inoue (2018): *Exploiting answer set programming with external sources for meta-interpretive learning*. Theory and Practice of Logic Programming 18(3-4), pp. 571–588, doi:10.1017/S1471068418000261.

[25] Henry Kautz (2022): *The third ai summer: Aaai robert s. engelmore memorial lecture*. AI Magazine 43(1), pp. 105–125, doi:10.1002/aaai.12036.

[26] Phuc H. Le-Khac, Graham Healy & Alan F. Smeaton (2020): *Contrastive Representation Learning: A Framework and Review*. IEEE Access 8, pp. 193907–193934, doi:10.1109/ACCESS.2020.3031549.

[27] Cheng Luo, Wei Liu, Jieyu Lin, Jiajie Zou, Ming Xiang & Nai Ding (2022): *Simple but Challenging: Natural Language Inference Models Fail on Simple Sentences*. In Yoav Goldberg, Zornitsa Kozareva & Yue Zhang, editors: Findings of the Association for Computational Linguistics: EMNLP 2022, Association for Computational Linguistics, pp. 3449–3462, doi:10.18653/v1/2022.findings-emnlp.252.

[28] 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, p. 3753–3763, doi:10.5555/3327144.3327291.

[29] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum & Jiajun Wu (2019): *The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision*. *arXiv preprint arXiv:1904.12584*, doi:10.48550/arXiv.1904.12584.

[30] Tom McCoy, Ellie Pavlick & Tal Linzen (2019): *Right for the Wrong Reasons: Diagnosing Syntactic Heuristics in Natural Language Inference*. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 3428–3448, doi:10.18653/v1/P19-1334.

[31] Arindam Mitra, Peter Clark, Oyvind Tafjord & Chitta Baral (2019): *Declarative question answering over knowledge bases containing natural language text with answer set programming*. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 01, pp. 3003–3010, doi:10.1609/aaai.v33i01.33013003.

[32] Stephen Muggleton (1991): *Inductive logic programming*. *New generation computing* 8, pp. 295–318, doi:10.1007/BF03037089.

[33] Stephen Muggleton & Luc De Raedt (1994): *Inductive logic programming: Theory and methods*. *The Journal of Logic Programming* 19, pp. 629–679, doi:10.1016/0743-1066(94)90035-3.

[34] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue & Ashwin Srinivasan (2012): *ILP turns 20: biography and future challenges*. *Machine learning* 86, pp. 3–23, doi:10.1007/s10994-011-5259-2.

[35] Nikita Nangia & Samuel R. Bowman (2019): *Human vs. Muppet: A Conservative Estimate of Human Performance on the GLUE Benchmark*. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4566–4575, doi:10.18653/v1/P19-1449.

[36] Hyun Oh Song, Yu Xiang, Stefanie Jegelka & Silvio Savarese (2016): *Deep metric learning via lifted structured feature embedding*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4004–4012, doi:10.1109/CVPR.2016.434.

[37] Ali Payani & Faramarz Fekri (2019): *Inductive logic programming via differentiable deep neural logic networks*. *arXiv preprint arXiv:1906.03523*, doi:10.48550/arXiv.1906.03523.

[38] Dhruva Pendharkar, Kinjal Basu, Farhad Shakerin & Gopal Gupta (2022): *An asp-based approach to answering natural language questions for texts*. *Theory and Practice of Logic Programming* 22(3), pp. 419–443, doi:10.1007/978-3-030-05998-9_4.

[39] Aarne Ranta (2004): *Grammatical framework*. *Journal of Functional Programming* 14(2), pp. 145–189, doi:10.1017/S0956796803004738.

[40] Mohammad Sadegh Rasooli & Joel R. Tetreault (2015): *Yara Parser: A Fast and Accurate Dependency Parser*. *arXiv preprint arXiv:1503.06733v2*, doi:10.48550/arXiv.1503.06733.

[41] 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*, pp. 3982–3992, doi:10.18653/v1/D19-1410.

[42] Raymond Reiter (1981): *On closed world data bases*. In: *Readings in artificial intelligence*, Elsevier, pp. 119–140, doi:10.1016/B978-0-934613-03-3.50014-3.

[43] Shachar Rosenman, Alon Jacovi & Yoav Goldberg (2020): *Exposing Shallow Heuristics of Relation Extraction Models with Challenge Data*. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pp. 3702–3710, doi:10.18653/v1/2020.emnlp-main.302.

[44] Prithviraj Sen, Breno WSR de Carvalho, Ryan Riegel & Alexander Gray (2022): *Neuro-symbolic inductive logic programming with logical neural networks*. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 36, pp. 8212–8219, doi:10.1609/aaai.v36i8.20795.

[45] Yibin Shen, Qianying Liu, Zhuoyuan Mao, Fei Cheng & Sadao Kurohashi (2022): *Textual Enhanced Contrastive Learning for Solving Math Word Problems*. In: *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 4297–4307, doi:10.18653/v1/2022.findings-emnlp.316.

[46] Koustuv Sinha, Prasanna Parthasarathi, Joelle Pineau & Adina Williams (2021): *UnNatural Language Inference*. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 7329–7346, doi:10.18653/v1/2021.acl-long.569.

[47] Shane Storks, Qiaozi Gao & Joyce Y. Chai (2020): *Recent Advances in Natural Language Inference: A Survey of Benchmarks, Resources, and Approaches*. *arXiv preprint arXiv:1904.11694*, doi:10.48550/arXiv.1904.01172.

[48] Vladimir N Vapnik & A Ya Chervonenkis (2015): *On the uniform convergence of relative frequencies of events to their probabilities*. In: *Measures of complexity: festschrift for alexey chervonenkis*, Springer, pp. 11–30, doi:10.1007/978-3-319-21852-6_3.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser & Illia Polosukhin (2017): *Attention is all you need*. *Advances in neural information processing systems* 30, doi:10.48550/arXiv.2002.05709.

[50] Adina Williams, Nikita Nangia & Samuel Bowman (2018): *A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference*. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122, doi:10.18653/v1/N18-1101.

[51] Adina Williams, Nikita Nangia & Samuel Bowman (2018): *A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference*. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1112–1122, doi:10.18653/v1/N18-1101.

[52] Yuanmeng Yan, Rumei Li, Sirui Wang, Fuzheng Zhang, Wei Wu & Weiran Xu (2021): *ConSERT: A Contrastive Framework for Self-Supervised Sentence Representation Transfer*. In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 5065–5075, doi:10.18653/v1/2021.acl-long.393.

[53] Jing Zhang, Bo Chen, Lingxi Zhang, Xirui Ke & Haipeng Ding (2021): *Neural, symbolic and neural-symbolic reasoning on knowledge graphs*. *AI Open* 2, pp. 14–35, doi:10.1016/j.aiopen.2021.03.001.

Architecture for Simulating Behavior Mode Changes in Norm-Aware Autonomous Agents

Sean Glaze
Miami University
Ohio, USA
glazesc@miamioh.edu

Daniela Inclezan
Miami University
Ohio, USA
inclezd@miamioh.edu

This paper presents an architecture for simulating the actions of a norm-aware intelligent agent whose behavior with respect to norm compliance is set, and can later be changed, by a human controller. Updating an agent's behavior mode from a norm-abiding to a riskier one may be relevant when the agent is involved in time-sensitive rescue operations, for example. We base our work on the *Authorization and Obligation Policy Language* $\mathcal{AOPL}$ designed by Gelfond and Lobo for the specification of norms. We introduce an architecture and a prototype software system that can be used to simulate an agent's plans under different behavior modes that can later be changed by the controller. We envision such software to be useful to policy makers, as they can more readily understand how agents may act in certain situations based on the agents' attitudes towards norm-compliance. Policy makers may then refine their policies if simulations show unwanted consequences.

1 Introduction

This paper introduces an architecture for simulating the actions to be taken by an intelligent agent that is aware of norms (i.e., policies[1]) governing the domain in which it acts. We assume that different agents may exhibit different behavior modes with respect to norm-compliance: some may be very cautious and norm-abiding, while others may exhibit a riskier behavior. We consider the case in which the behavior mode under which an agent operates is set by a human controller who can update it if needed, for instance in cases when the agent is involved in a time-sensitive rescue operation.

This architecture is relevant to modeling physical intelligent agents that act autonomously, for instance robots deployed in harsh environments (underwater, on Mars, in mines), and whose settings may be re-adjusted by a human controller if the circumstances require it, but this is done sparingly in emergency situations. The architecture is also crucial to simulating the behavior of human agents with different norm-abiding attitudes, especially if such attitudes change over time. This can be of value to policy makers as testing their policies on different human agent models can lead to policy improvement, if unwanted consequences are observed in the simulation (similarly to work by Corapi et al. [4] on creating *use cases* for policy development and refinement).

In our work, we utilize the *Authorization and Obligation Policy Language* ($\mathcal{AOPL}$) by Gelfond and Lobo [8] for norm specification, due to its close connection to Answer Set Programming (ASP). In fact, the semantics of $\mathcal{AOPL}$ and the notion of norm-compliance are defined via a translation into ASP. This allows us to leverage existing ASP methodologies for representing dynamic domains, planning, and creating agent architectures, as well as ASP solvers like CLINGO (`https://potassco.org/clingo/`) or DLV (`https://www.dlvsystem.it/dlvsite/`). A reason for using $\mathcal{AOPL}$ instead of representing norms directly in ASP (as soft constraints for example) is that $\mathcal{AOPL}$ provides policy analysis capabilities [10], which are important for checking that a policy imposed on an agent is actually valid and

[1] We use the words *norms* and *policy* interchangeably in this paper.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 95–107, doi:10.4204/EPTCS.416.7

unambiguous; similar policy analysis would be difficult to conduct without the use of a higher-level language for norm specification. In addition to $\mathcal{AOPL}$, we build upon work on norm-aware autonomous agents by [9] who introduced the notion of behavior modes with respect to norm-compliance.

Our main contributions are two-fold: (1) we **introduce an architecture** for norm-aware autonomous agents who may exhibit different behavior modes and may experience changes between bahavior modes; and (2) we **implement a software system** for the simulation of an agent's actions under behavior modes that may change over time.

In the remainder of the paper, we start with background information in Section 2 and provide a motivating example in Section 3. We introduce our architecture in Section 4, then our software system for simulation in Section 5, and examine the system's evaluation in Section 6. We discuss related work in Section 7 and end with conclusions in Section 8.

2 Background

In this section we introduce the norm-specification language $\mathcal{AOPL}$ and behavior modes for norm-aware agents. We assume that readers are familiar with ASP and otherwise direct them to outside resources on ASP [7, 11, 6].

2.1 Norm-Specification Language $\mathcal{AOPL}$

Gelfond and Lobo [8] introduced the *Authorization and Obligation Policy Language* ($\mathcal{AOPL}$) for specifying policies for an intelligent agent acting in a dynamic environment. A policy is a collection of authorization and obligation statements. An *authorization* indicates whether an agent's action is permitted or not, and under which conditions. An *obligation* describes whether an agent is obligated or not obligated to perform a specific action under certain conditions. An $\mathcal{AOPL}$ policy works in conjunction with a dynamic system description of the agent's environment written in an action language such as $\mathcal{AL}_d$ [5]. The signature of the dynamic system description includes predicates denoting *sorts* for the elements in the domain; *fluents* (i.e., properties of the domain that may be changed by actions); and *actions*. An $\mathcal{AL}_d$ system description defines the domain's transition diagram whose states are complete and consistent sets of fluent literals and whose arcs are labeled by action atoms (shortly *actions*).

The signature of an $\mathcal{AOPL}$ **policy** includes the signature of the associated dynamic system and additional predicates *permitted* for authorizations, *obl* for obligations, and *prefer* for specifying preferences between authorizations or obligations. A *prefer* atom is created from the predicate *prefer*; similarly for *permitted* and *obl* atoms.

An $\mathcal{AOPL}$ **policy** $\mathcal{P}$ is a finite collection of statements of the form:

$$
\begin{array}{rlll}
 & permitted\,(e) & \textbf{if } cond & (1a)\\
 & \neg permitted\,(e) & \textbf{if } cond & (1b)\\
 & obl\,(h) & \textbf{if } cond & (1c)\\
 & \neg obl\,(h) & \textbf{if } cond & (1d)\\
d : \textbf{normally} & permitted(e) & \textbf{if } cond & (1e)\\
d : \textbf{normally} & \neg permitted(e) & \textbf{if } cond & (1f)\\
d : \textbf{normally} & obl(h) & \textbf{if } cond & (1g)\\
d : \textbf{normally} & \neg obl(h) & \textbf{if } cond & (1h)\\
 & prefer(d_i, d_j) & & (1i)
\end{array}
$$

where e is an elementary action; h is a happening (i.e., an elementary action or its negation[2]); *cond* is a set of atoms of the signature, except for atoms containing the predicate *prefer*; d in (1e)-(1h) and d_i, d_j in (1i) denote defeasible rule labels. Rules (1a)-(1d) encode *strict* policy statements, while rules (1e)-(1h) encode *defeasible* statements. Rule (1i) captures *priorities* between defeasible statements.

The **semantics** of an $\mathcal{AOPL}$ policy determine a mapping $\mathcal{P}(\sigma)$ from states of a transition diagram $\mathcal{T}$ into a collection of *permitted* and *obl* literals. To formally describe the semantics of $\mathcal{AOPL}$, a translation of a policy $\mathscr{P}$ and a state σ of the transition diagram into ASP is defined as $lp(\mathscr{P},\sigma)$ as described in the paper by Gelfond and Lobo [8]. Properties of an $\mathcal{AOPL}$ policy $\mathcal{P}$ are defined in terms of the answer sets of the logic program $lp(\mathscr{P},\sigma)$ expanded with appropriate rules.

The following definitions by Gelfond and Lobo are relevant to our work (original definition numbers in parenthesis). In what follows a denotes a (possibly) compound action (i.e., a set of simultaneously executed elementary actions), while e refers to an elementary action. An event $\langle\sigma,a\rangle$ is a pair consisting of a state σ and an action a executed in σ.

Definition 1 (Consistency and Categoricity – Defs. 3 and 6) *A policy $\mathcal{P}$ for $\mathcal{T}$ is called* consistent *if for every state σ of $\mathcal{T}$, the logic program $lp(\mathcal{P},\sigma)$ has an answer set. It is called* categorical *if $lp(\mathcal{P},\sigma)$ has exactly one answer set.*

Definition 2 (Policy Compliance for Authorizations and Obligations – Defs. 4, 5, and 9) • *An event $\langle\sigma,a\rangle$ is* strongly-compliant *with authorization policy $\mathcal{P}$ if for every $e \in a$ the logic program $lp(\mathcal{P},\sigma)$ entails $permitted(e)$.*
• *An event $\langle\sigma,a\rangle$ is* weakly-compliant *with authorization policy $\mathcal{P}$ if for every $e \in a$ the logic program $lp(\mathcal{P},\sigma)$ does not entail $\neg permitted(e)$.*
• *An event $\langle\sigma,a\rangle$ is* non-compliant *with authorization policy $\mathcal{P}$ if for every $e \in a$ the logic program $lp(\mathcal{P},\sigma)$ entails $\neg permitted(e)$.*
• *An event $\langle\sigma,a\rangle$ is* compliant *with obligation policy $\mathcal{P}$ if*
– *For every $obl(e) \in \mathcal{P}(\sigma)$ we have that $e \in a$, and*
– *For every $obl(\neg e) \in \mathcal{P}(\sigma)$ we have that $e \notin a$.*

2.2 Behavior Modes in Norm-Aware Autonomous Agents

Harders and Inclezan [9] introduced an ASP framework for plan selection for norm-aware autonomous agents, where norms were specified in $\mathcal{AOPL}$. They built upon observations by Inclezan [10] indicating that, for categorical $\mathcal{AOPL}$ policies, all strongly-compliant actions are also weakly-compliant w.r.t. authorizations and that modality conflicts between authorizations and obligations may occur when the $\mathcal{AOPL}$ policy simultaneously contains obligations and prohibitions to execute an action. Instead, the notion of an *underspecified* event was introduced to denote an event that is not explicitly known to be compliant nor non-compliant w.r.t. authorizations, and a *modality ambiguous* event as an event arising from a modality conflict. Harders and Inclezan proposed that agents may have different attitudes towards norm compliance that would impact the selection of the "best" plan. They called these attitudes *behavior modes* and introduced different metrics that can be used to express them. They also presented some predefined agent behavior modes, defined as follows: (a) **Safe Behavior Mode** – prioritizes events that are explicitly known to be compliant and does not execute non-compliant actions; (b) **Normal Behavior Mode** – prioritizes plan length and then actions explicitly known to be compliant, while not executing non-compliant actions; and (c) **Risky Behavior Mode** – disregards policy rules, but does not go out of its way to break rules either.

[2] If $obl(\neg e)$ is true, then the agent must not execute e.

3 Example

For illustration purposes, consider a *Mining Domain* consisting of a 3x3 square grid of locations with an associated risk level (low, medium, or high) and three ores (gold, silver, and iron) with unique locations across the grid. The mining robot can collect ores or move between adjacent locations. The mining robot's goal is to collect all three ores. The norm that is imposed in this domain is that the collection of ores must happen in the sequence: gold first, then silver, and finally iron.

The mining robot has three *behavior modes*: Safe, Normal, and Risky, as defined in Section 2.2, but expanded with some additional policies. The Safe agent is obligated to move only through low-risk locations, the Normal agent is obligated to only move through low or medium-risk locations, and the Risky agent moves freely throughout the grid with no regard for the risk level of locations. Furthermore, as the Risky behavior mode does not have any regard for policies, an agent in this mode will collect ores in whichever order leads to the shortest plan possible.

We will now discuss a specific scenario within the mining domain shown in Figure 1. In this illustration, locations are labeled l0 to l8, with connected locations indicated by a black line. Each location is colored green, yellow, or red to indicate a low, medium, or high-risk level, respectively. The mining robot is depicted in its initial location and the locations of ores are indicated by their corresponding labels in the periodic table.

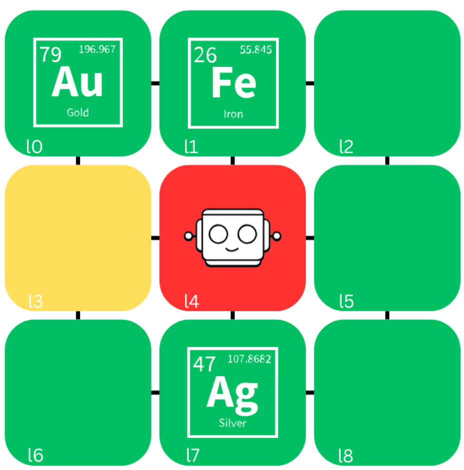


Figure 1: Mining Domain: Sample Scenario

Table 1: Plan with Behavior Mode Changes

*** Begin in Safe Mode ***
`0. Move from l4 to l1`
`1. Move from l1 to l0`
`2. Collect gold`
*** Change to Normal Mode ***
`3. Move from l0 to l3`
`4. Move from l3 to l6`
`5. Move from l6 to l7`
`6. Collect silver`
*** Change to Risky Mode ***
`7. Move from l7 to l4`
`8. Move from l4 to l1`
`9. Collect iron`

Table 2 shows the plans that the agent devises depending on its behavior mode. This scenario illustrates general outcomes, where cautious behavior modes result in longer plans, while the Risky behavior mode generates the shortest plans, with a trade-off of a detrimental effect on safety and policy-compliance. The longer plan devised in the Safe mode is caused by the inability to move through location l3, which is not a low-risk location. The Risky mining robot produces the shortest plan because it disregards location risk and the policy of collecting ores in a specific order.

Now let's consider behavior mode changes in this scenario. Table 1 shows the plan that is generated by a mining robot that begins in Safe mode, is switched by the controller to Normal mode at time step 3, and is switched again to Risky mode at time step 7. In case of emergency or changing priorities, a more risky behavior may be desired by the controller of such a robot, even though it does result in a higher degree of danger for the robot. We want our architecture to simulate such behavior mode modifications.

Table 2: Plans for the Scenario in Fig. 1 for Different Behavior Modes

Safe Behavior Mode	Normal Behavior Mode	Risky Behavior Mode
0. Move from l4 to l1	0. Move from l4 to l1	0. Move from l4 to l7
1. Move from l1 to l0	1. Move from l1 to l0	1. Collect silver
2. Collect gold	2. Collect gold	2. Move from l7 to l4
3. Move from l0 to l1	3. Move from l0 to l3	3. Move from l4 to l1
4. Move from l1 to l2	4. Move from l3 to l6	4. Collect iron
5. Move from l2 to l5	5. Move from l6 to l7	5. Move from l1 to l0
6. Move from l5 to l8	6. Collect silver	6. Collect gold
7. Move from l8 to l7	7. Move from l7 to l6	
8. Collect silver	8. Move from l6 to l3	
9. Move from l7 to l8	9. Move from l3 to l0	
10. Move from l8 to l5	10. Move from l0 to l1	
11. Move from l5 to l2	11. Collect iron	
12. Move from l2 to l1		
13. Collect iron		

4 Architecture

We identified three questions that needed to be answered during the development of this architecture, outlined below together with our design decisions:

- *How will an agent adjust its plan when its behavior mode is modified?*

  Design Decision: The agent will devise a new plan with its new behavior mode, starting at the time step that the behavior mode modification is set to take effect.

- *How will the agent's memory mechanism work with respect to already executed actions of a plan? In other words, how will the agent deal with prior actions that may not satisfy the definition of its new behavior mode?*

  Design Decision: The agent remembers the behavior mode under which it operated at each point in time and checks requirement satisfaction w.r.t. to the behavior mode settings in place when the action was executed, to mimic real world situations where new laws are not applied retrospectively.

- *Does the agent need to be aware that its behavior mode is liable to be modified at later points in time?*

  Design Decision: The agent is not explicitly aware that its behavior mode can be modified. However, we introduce the concept of *subgoals* so that the agent can strive to partially complete its overall goal if its current behavior mode prevent completing the goal as a whole.

The proposed architecture consists of two distinct components that work in conjunction: an ASP Component and a Python Component, discussed in detail in the following subsections. Figure 4 provides an overall view of the proposed architecture.

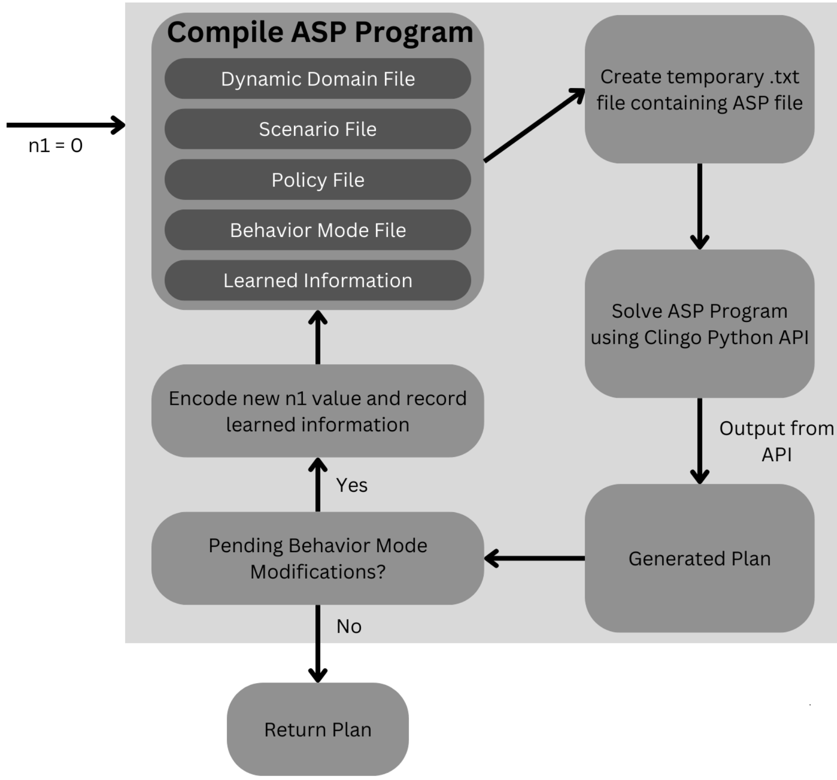


Figure 2: Illustration of Proposed Architecture

4.1 ASP Component

The ASP Component consists of five pieces: the dynamic domain encoding, scenario encoding, policy encoding, behavior mode encoding, and learned information.

The *dynamic domain encoding* contains the objects, statics, fluents, actions, and axioms that define the dynamic domain, encoded according to established ASP methodologies [6]. In the Mining Domain, the objects are *locations* and *ores*. Statics are used to describe whether two locations are connected, as well as the risk level of a location ($has\_risk\_level(l, level)$). Inertial fluents at play are the agent's location ($at\_loc(l)$); whether the agent possesses a certain ore or not ($has\_ore(o)$); and the locations of the ores ($ore\_loc(o,l)$). The agent can perform two actions: move from one location to another ($move(l_1,l_2)$); and collect an ore ($collect(o)$). It has an additional action *wait* that does not change the state of the domain.

The *scenario encoding* consists of a list of facts that are true at time step 0. For the Mining Domain, this means specifying the risk level of each of the locations, the initial location of the agent, and the location of the ores.

The *policy encoding* contains the ASP translations of the $\mathcal{AOPL}$ policies that govern the dynamic domain. For the Mining Domain, there is only one policy that applies by default: the agent is obligated to collect the ores in the sequence gold, silver, iron. This is encoded in two separate $\mathcal{AOPL}$ rules – one that says the agent is obligated not to collect silver unless it possesses gold and another that says the agent is obligated not to collect iron unless it possesses silver:

$$\begin{array}{lll} obl(\neg collect(silver)) & \textbf{if} & \neg has\_ore(gold) \\ obl(\neg collect(iron)) & \textbf{if} & \neg has\_ore(silver) \end{array}$$

The *behavior mode encoding* in this architecture considers tree behavior modes: Safe, Normal, and Risky. As previously mentioned, the exact definitions of these behavior modes are meant to be tailored to the dynamic domain's specific needs. For example, in the Mining Domain, the Safe and Normal agents are under additional policies. Specifically, the Safe agent is obligated not to move through high or medium-risk locations, and the Normal agent is obligated not to move through high-risk areas. These rules are written in $\mathcal{AOPL}$, as shown below for the Safe agent, and are translated into ASP to be used in our architecture:

$$obl(\neg move(L1,L2)) \quad \textbf{if} \quad has\_risk\_level(L_2,high)$$
$$obl(\neg move(L1,L2)) \quad \textbf{if} \quad has\_risk\_level(L_2,medium)$$

The *behavior mode encoding* also contains general ASP rule for planning, such as:

```
1 {occurs(A,I) : action(A)} 1:- step(I), I >= n1.
```

This rule says that at each time step $I \geq n1$, the agent must perform exactly one action. The constant, $n1$, is an integral part of this architecture: it represents the time step in which the new behavior mode is to take effect. For example, if we are in a scenario where we want the agent's initial behavior mode to be b_0 and switch to b_1 at time step i, then $n1 = 0$ for each time step $t < i$, and $n1 = i$ for each time step $t \geq i$. This ensures that only the planned actions at time steps greater than or equal to i have to obey the definition of behavior mode b_1.

In each of the behavior mode's encodings, we additionally have several metrics that are calculated and considered by the agent when devising its plan, as in work by [9]. What differentiates each of the behavior modes is the priority that is given to each of the metrics in the planning process, as described in Section 2.2. For example, in the Safe behavior mode's encoding, we see the following ASP rule:

```
#maximize{ N4@4 : subgoal_count(N4);
           N3@3 : percentage_strongly_compliant(N3);
           N2@2 : percentage_underspecified(N2);
           N1@1 : wait_count(N1)}.
```

This says that the metric `subgoal_count` should be prioritized first, `percentage_strongly_compliant` should be prioritized second, `percentage_underspecified` should be prioritized third, and `wait_count` should be prioritized last. The `subgoal_count` metric is a count of the number of subgoals that the agent completes during the plan. This is a novel inclusion in our proposed architecture. In the Mining Domain, the maximum number of subgoals that the agent can complete is three, one subgoal corresponding to the collection of each of the ores. The ASP encoding for this is:

```
subgoal(has_ore(gold)). subgoal(has_ore(silver)). subgoal(has_ore(iron)).
subgoal_count(N) :- #count{F : subgoal(F), holds(F, n)} = N.
```

The `percentage_strongly_compliant` and `percentage_underspecified` metrics come from work by Harders and Inclezan [9]. Recall that a *strongly-compliant* action is one that is explicitly permitted by the agent's policies and an *underspecified* action is one that is neither permitted nor not permitted by the agent's policies. The safe agent prioritizes actions that are explicitly permitted, because it is designed to act in an extremely cautious way, even if unnecessary. Finally, the `wait_count` metric is a count of the number of *wait* actions in the agent's plan. The higher the `wait_count`, the shorter the plan. Agents under this proposed architecture only perform waiting actions after they have completed as many subgoals as possible. This is encoded in ASP as:

```
:- occurs(wait, I1), occurs(A, I2), I2 > I1, I1 >= n1, A != wait.
```

Now that we have an understanding of what each of these metrics represent, let's compare the prioritization order of the Safe agent to that of the Normal agent.

```
#maximize{ N4@4 : subgoal_count(N4);
           N3@3 : wait_count(N3);
           N2@2 : percentage_underspecified(N2);
           N1@1 : percentage_strongly_compliant(N1)}.
```

The Normal agent still prioritizes first completing as many subgoals as possible, but instead of also trying to maximize the number of strongly-compliant actions in the plan, it values a shorter plan. Additionally, both the Safe and Normal agent behavior modes have constraints saying that no non-compliant actions w.r.t. obligations are allowed. The Risky agent only considers two metrics in its planning process, `subgoal_count` and `wait_count`, in that order. This allows the Risky agent to devise the shortest plan possible while completing as many subgoals as possible, with the trade-off that it completely disregards any policies that are imposed on it. The ASP encoding is:

```
#maximize{ N2@2: subgoal_count(N2); N1@1: wait_count(N1)}.
```

Finally, the *learned information* refers to the facts formed by *holds* and *occurs* literals that are true prior to the time step when the behavior mode modification took effect.

4.2 Python Component

The Python Component of the proposed architecture is what allows us to manage the behavior mode modification process. This component utilizes the CLINGO Python API, which allows developers to solve ASP programs and analyze their output using Python code. For the Mining Domain, we present a class called `MiningDomainSolver`, which takes as input a scenario number, an initial behavior mode, and a list of behavior mode changes and the time steps when they are to take effect. Once this class is instantiated, a user may call the class's function called `generate_plan_with_bmode_changes()`, which returns the plan as a string. This function follows the control flow outlined below. It is also worth noting that this control flow is not specific to the Mining Domain, and can be applied to any other dynamic domain under this proposed architecture:

1. $n1$ is computed. As mentioned previously, $n1 = 0$, when we are solving the ASP program corresponding to the initial behavior mode, and $n1$ is equal to the time step of each behavior mode change after that.
2. The ASP program is created inside of a string variable by reading the contents of the text files of the dynamic domain (i.e., the dynamic domain encoding, scenario encoding, policy encoding, and behavior mode encoding). *Learned information* (stored inside of a string variable) is also added to the ASP program.
3. A temporary text file is created and the ASP program is written to it.
4. A CLINGO *control* object is created using the CLINGO API, and the temporary file is loaded into it using its provided `load()` function.
5. The ASP program is solved using the `solve()` function of the *control* object. This function solves the loaded ASP program and outputs the literals in the answer set that are specified using CLINGO's `#show` directive in the ASP program.
6. If there is a behavior mode change, these literals are saved to a class variable and filtered to produce the *learned information* for the next iteration.

5 Software System

Next, let us discuss the graphical user interface (GUI) that was developed as a proof of concept for a program that allows a controller to make behavior mode modifications of an agent. The software is available at `https://github.com/scglaze/MiningRobotDomainGUI`.

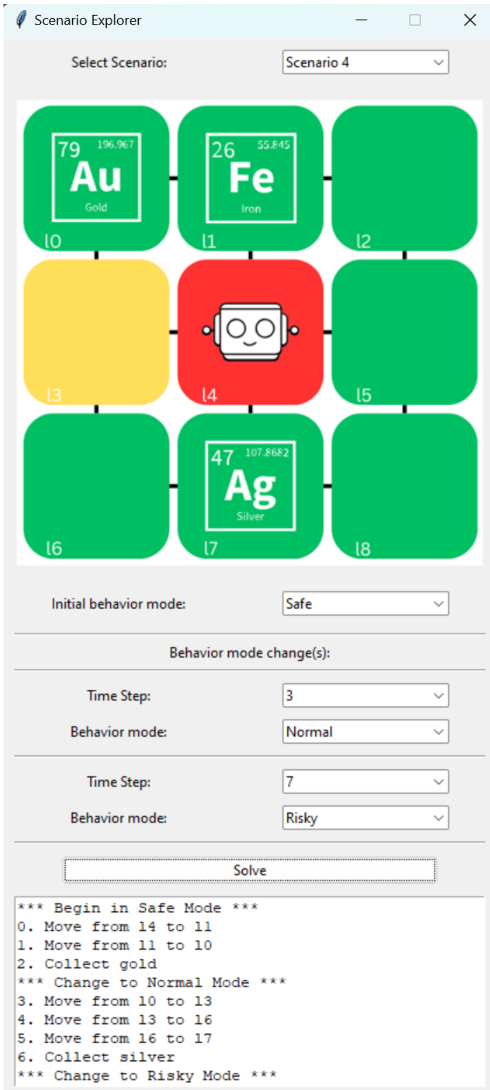


Figure 3: GUI screenshot with input parameters for the Mining Domain Scenario 4

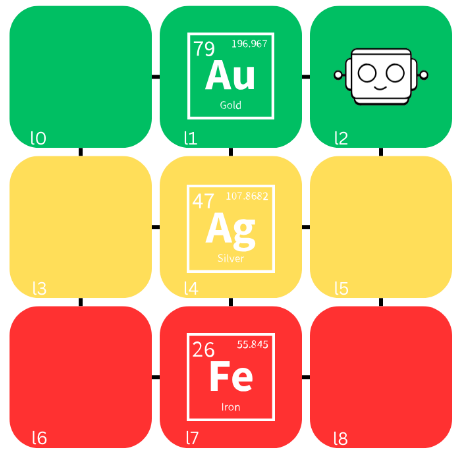


Figure 4: Mining Domain Scenario 9

We leveraged the Tkinter Python library for GUI development. The GUI allows a user to select one of the 10 scenarios that we have prepared from the Mining Domain, an initial behavior mode, and up to two behavior mode changes. When the user selects a scenario, a graphic for that scenario appears to serve as a visual aid. Once the user is finished inputting their desired parameters, there is a "Solve" button that initiates the solving process. This solving process is performed by the aforementioned `MiningDomainSolver` class described in the previous section, by feeding the user's input to it as its parameter. Before this is done, though, there are several validation checks that are performed. For example, the user must input both a behavior mode *and* a time step for each behavior mode modification. If there are any validation checks that are violated, then a dialog box appears with a description of the error. If there are no validation errors, once the solving process is finished, the generated plan is displayed in a user-friendly manner in a text box at the bottom of the GUI, as shown in Figure 3.

6 Evaluation

Runtime Performance: We ran experiments on the 10 scenarios in the Mining Domain. We measured the runtime performance for each of the three behavior modes by themselves, and for the six combinations of first-order behavior mode modifications (i.e. only one modification made during the plan). We varied the time step when the modification occurred from scenario to scenario, based on what we subjectively deemed as leading to most illustrative changes in plans. Additionally, we measured the runtime performance of second-order behavior mode modifications for two of the scenarios that are more complex. We present the runtime performance of Scenario #4 from Figure 3 in Table 3 and Scenario #9 from Figure 4 in Table 4. Time steps when behavior mode modifications are made are listed in parenthesis. Scenario #9 involves second-order behavior mode modification. The runtime performance that we report does not come from the CLINGO solver itself, but instead, is measured via code that was integrated into the Python component for this experiment. This test code utilizes the `time` Python library. The reason we went this route instead of measuring the reported runtime from CLINGO, is that our proposed architecture requires an additional `solve()` for each behavior mode modification that is made. We ran each experiment 10 times, and report the mean in seconds (**T (s)**) and standard deviation (**SD**). All experiments were performed on a machine with an Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz 1.50 GHz processor and 16 GB RAM.

Table 3: Mining Domain Scenario 4: Runtime Data (Python + CLINGO)

Behavior Modes	**T (s)**	**SD**
Safe	1.684	0.12
Normal	1.379	0.08
Risky	0.181	0.08
Safe to Normal (2)	2.641	0.11
Safe to Risky (2)	1.881	0.06
Normal to Safe (2)	2.666	0.14
Normal to Risky (2)	1.764	0.24
Risky to Safe (2)	1.452	0.17
Risky to Normal (2)	0.944	0.07

Table 4: Mining Domain Scenario 9: Runtime Data (Python + CLINGO)

Behavior Modes	**T (s)**	**SD**
Safe	0.652	0.04
Normal	0.565	0.07
Risky	0.094	0.01
Safe to Normal (2)	1.059	0.08
Safe to Risky (2)	0.767	0.04
Normal to Safe (2)	0.963	0.12
Normal to Risky (2)	0.596	0.03
Risky to Safe (2)	0.516	0.03
Risky to Normal (2)	0.481	0.04
Safe to Normal (2) to Risky (4)	1.202	0.12
Safe to Normal (3) to Risky (6)	1.090	0.12

We see that the Safe behavior mode has a slightly longer runtime than that of the Normal behavior mode, and that the Risky behavior mode has the shortest runtime overall, which is a trend observed across all 10 scenarios. This is intuitive, as the Safe behavior mode takes the most amount of factors (i.e., aggregates) into consideration during plan generation, and the Risky behavior mode takes significantly less factors into consideration. Similarly, we observe that the runtime of behavior mode *modifications* (with the time step when the modification occurs specified in parenthesis) follows this same trend – modifications involving the Risky mode add less runtime than either of the other behavior modes, and the Safe mode adds the most to runtime. Scenario #9 additionally tests second-order behavior mode modifications. We selected modifying the agent's behavior mode from Safe, to Normal, to Risky because the agent begins in a safe area of the grid, where the gold is also located. The silver is located adjacent to the gold but in a medium-risk location that the Safe agent cannot access. Therefore, the Safe agent will *wait* indefinitely, unless there is a behavior mode modification. This behavior is mirrored by the Normal agent after it collects the silver. Hence, the plan generated by the agent with behavior mode parameters

"Safe to Normal (2) to Risky (4)" is set up to collect the ores without the Safe or Normal agents waiting at all, and the agent with behavior mode parameters "Safe to Normal (3) to Risky (6)" is set up for the Safe and Normal agents to wait for exactly 1 time step before its behavior mode is modified to a more Risky one that allows them to complete another subgoal. An interesting observation is that the agent with behavior mode parameters "Safe to Normal (3) to Risky (6)" has a faster runtime than that with parameters "Safe to Normal (2) to Risky (4)." We speculate that this is because the plan that is generated by the Normal agent at time step 3 has a shorter span of time steps to plan for than when starting at time step 2, and likewise for the Risky agent at time step 6 versus 4.

We also ran experiments on the 14 scenarios of the *Room Domain* by Harders and Inclezan [9]. The results are in Table 5 and they match the observations for the Mining Domain.

Table 5: Performance Results: Room Domain (Python + CLINGO)

	Safe Mode		**Normal Mode**		**Risky Mode**		**One Behavior Mode Change**		
Scen. #	**T (s)**	**SD**	**T (s)**	**SD**	**T (s)**	**SD**	**T (s)**	**SD**	**Change**
1	6.876	0.38	7.468	0.21	7.196	0.20	14.621	0.34	Safe to Normal (1)
2	7.777	0.50	7.620	0.56	8.521	2.40	15.541	2.31	Risky to Safe (2)
3	7.772	0.14	9.667	2.95	7.689	0.48	14.345	0.21	Safe to Risky (3)
4	7.763	0.21	7.644	0.24	7.474	0.55	14.271	0.37	Risky to Safe (1)
5	7.316	0.15	7.190	0.06	7.085	0.09	14.096	0.23	Risky to Normal (1)
6	7.762	0.40	7.232	0.17	7.159	0.18	13.795	0.19	Safe to Normal (2)
7	7.836	0.38	7.442	0.31	7.223	0.18	13.932	0.48	Risky to Normal (2)
8	7.196	0.09	7.487	0.10	7.859	0.68	14.135	0.24	Safe to Risky (2)
9	9.103	0.12	7.727	0.18	7.665	0.20	15.821	0.39	Safe to Risky (2)
10	7.305	0.15	7.307	0.17	7.207	0.11	13.871	0.36	Normal to Safe (2)
11	8.065	0.41	7.870	0.21	7.706	0.22	14.882	0.27	Normal to Risky (1)
12	7.741	0.17	7.689	0.42	7.488	0.09	14.520	0.24	Normal to Safe (2)
13	7.762	0.30	7.712	0.21	7.739	0.35	14.720	0.13	Safe to Normal (2)
14	8.369	0.27	7.395	0.16	7.334	0.11	13.346	0.44	Normal to Safe (4)

GUI Usability Study: The final evaluation was a usability study for the GUI that we presented in Section 5. Our participants ($N = 6$) were given a brief explanation of the Mining Domain, and necessary background information on ASP planning. Then, they were asked to download and run an executable file for the GUI seen in Figure 3, and to test all 10 scenarios with different behavior mode parameters. Finally, they answered questions on a 5-point Likert scale. The average score and standard deviation for each question's responses are reported in Table 6. While scores were generally high, especially for question 6, we do note the lower scores for questions 1, 2, 8. This indicates that the prototype GUI can be improved by using more modern-looking widgets, facilitating the process of downloading it, and providing more descriptive error messages that are displayed when input validation checks that are violated.

7 Related Work

Our work expands on Harders and Inclezan's [9] notions of behavior modes w.r.t. norm-compliance. Another work on norm-aware agents is that by by Meyer and Inclezan [12] who created the $\mathcal{APIA}$ architecture for norm-aware *intentional* agents. $\mathcal{APIA}$ agents operate with *activities* instead of simple

Table 6: Usability Study Results

	Survey Question	Average Score (scale 1-5)	SD
1	The executable (.exe) file was easy to download and run.	3.83	1.47
2	The GUI has a nice look and feel.	3.60	0.89
3	The GUI was easy to interact with.	4.67	0.52
4	I did not encounter any odd behavior from the GUI.	4.83	0.41
5	The images depicting the different scenarios were a useful resource for understanding the generated plan.	4.50	0.55
6	It was easy to change behavior modes.	5.00	0.00
7	I understand the plan that was generated by the program.	4.50	0.84
8	Error messages were easy to understand (Only answer this question if you received error messages).	3.67	1.15

plans, by building upon the AIA architecture by [3]. $\mathcal{APIA}$ agents can reason about agent intentions, but does not allow the agent's controller to easily set and change behavior modes. [14] introduced an ASP framework for reasoning and planning with norms for autonomous agents. The agent actions in their framework have an associated duration and can incur penalties, while policies have an expiration deadline. On the other hand, their framework does not model different behavior modes and changes between behavior modes, which is the focus of this paper. Other existing approaches on norm-aware agents focus solely on compliant behavior (e.g., [13, 1]), while we were interested in studying a range of behavior modes on a spectrum for norm-abiding to non-compliant to enable the simulation of human behavior as well. In our work, we assume that changes between behavior modes are justified in certain situations, such as emergency rescue operations, and this should be modeled and simulated. The question of emergency situations in relation to norms was previous studied by Alves and Fernández [2], but only in the context of access control policies. In contrast, the use $\mathcal{AOPL}$ for norm specification in our architecture allows us to express not only access control policies (i.e., authorizations), but also obligations, both strict and defeasible, and preferences between policy statements. In terms of defining behavior modes via priorities between different metrics, our work indicates some connections to Son and Pontelli's $\mathcal{PP}$ for specifying basic preferences [15]. It is not clear though whether maximizations of percentage metrics, which occur in our description of behavior modes, can be achieved within the $\mathcal{PP}$ framework.

8 Conclusions and Future Work

We presented an ASP framework that defines how the controller of norm-aware autonomous agents can modify their behavior modes under the plan-choosing framework proposed by [9]. We introduced a Python component that includes a wrapper class that can be used as the behavior mode-changing mechanism and a GUI with the potential for generalization, for other domains as well.

In the future, one could generalize this proposed ASP simulation framework so that a controller can manipulate multiple agents' behavior modes as they work toward achieving their goal(s). Optimizing ASP encodings and Python code is another future goal, as it would allow for larger and more complicated dynamic domains to be simulated. Finally, one could continue to develop additional behavior modes, outside of the three that are used in this framework. This would allow for more nuanced agent behavior to be modeled under this framework and generally in ASP.

References

[1] Natasha Alechina, Mehdi Dastani & Brian Logan (2012): *Programming norm-aware agents*. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '12, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, p. 1057–1064, doi:10.5555/2343776.2343848.

[2] Sandra Alves & Maribel Fernandez (2017): *A graph-based framework for the analysis of access control policies*. *Theoretical Computer Science* 685, pp. 3–22, doi:10.1016/j.tcs.2016.10.018.

[3] Justin Blount, Michael Gelfond & Marcello Balduccini (2015): *A Theory of Intentions for Intelligent Agents - (Extended Abstract)*. In: *Proceedings of the 13th International Conference on Logic Programming and Nonmonotonic Reasoning*, *LNCS* 9345, Springer, pp. 134–142, doi:10.1007/978-3-319-23264-5_12.

[4] Domenico Corapi, Marina De Vos, Julian A. Padget, Alessandra Russo & Ken Satoh (2010): *Norm Refinement and Design through Inductive Learning*. In Marina De Vos, Nicoletta Fornara, Jeremy V. Pitt & George A. Vouros, editors: *Coordination, Organizations, Institutions, and Norms in Agent Systems VI - COIN 2010 International Workshops*, *Lecture Notes in Computer Science* 6541, Springer, pp. 77–94, doi:10.1007/978-3-642-21268-0_5.

[5] Michael Gelfond & Daniela Inclezan (2013): *Some properties of system descriptions of ALd*. *J. Appl. Non Class. Logics* 23(1-2), pp. 105–120, doi:10.1080/11663081.2013.798954.

[6] Michael Gelfond & Yulia Kahl (2014): *Knowledge Representation, Reasoning, and the Design of Intelligent Agents*. Cambridge University Press, doi:10.1017/CBO9781139342124.

[7] Michael Gelfond & Vladimir Lifschitz (1991): *Classical Negation in Logic Programs and Disjunctive Databases*. *New Generation Computing* 9(3/4), pp. 365–386, doi:10.1007/BF03037169.

[8] Michael Gelfond & Jorge Lobo (2008): *Authorization and Obligation Policies in Dynamic Systems*. In Maria Garcia de la Banda & Enrico Pontelli, editors: *Logic Programming*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 22–36, doi:10.1007/978-3-540-89982-2_7.

[9] Charles Harders & Daniela Inclezan (2023): *Plan Selection Framework for Policy-Aware Autonomous Agents*. In Sarah Alice Gaggl, Maria Vanina Martinez & Magdalena Ortiz, editors: *Logics in Artificial Intelligence - 18th European Conference, JELIA*, *LNCS* 14281, Springer, pp. 638–646, doi:10.1007/978-3-031-43619-2_43.

[10] Daniela Inclezan (2023): *An ASP Framework for the Refinement of Authorization and Obligation Policies*. *Theory and Practice of Logic Programming* 23(4), p. 832–847, doi:10.1017/S147106842300011X.

[11] Victor W. Marek & Miroslaw Truszczynski (1999): *Stable Models and an Alternative Logic Programming Paradigm*. In Krzysztof R. Apt, Victor W. Marek, Mirek Truszczynski & David Scott 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.

[12] John Meyer & Daniela Inclezan (2021): *APIA: An Architecture for Policy-Aware Intentional Agents*. In: *Proceedings of the 37th International Conference on Logic Programming (Technical Communications)*, *EPTCS* 345, pp. 84–98, doi:10.4204/EPTCS.345.23.

[13] Nir Oren, Wamberto Vasconcelos, Felipe Meneguzzi & Michael Luck (2011): *Acting on Norm Constrained Plans*. In João Leite, Paolo Torroni, Thomas Ågotnes, Guido Boella & Leon van der Torre, editors: *Computational Logic in Multi-Agent Systems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 347–363, doi:10.1007/978-3-642-22359-4_24.

[14] Zohreh Shams, Marina De Vos, Julian A. Padget & Wamberto Weber Vasconcelos (2017): *Practical reasoning with norms for autonomous software agents*. *Eng. Appl. Artif. Intell.* 65, pp. 388–399, doi:10.1016/J.ENGAPPAI.2017.07.021.

[15] Tran Cao Son & Enrico Pontelli (2006): *Planning with preferences using logic programming*. *Theory and Practice of Logic Programming* 6(5), p. 559–607, doi:10.1017/S1471068406002717.

Policies, Penalties, and Autonomous Agents (Extended Abstract)

Vineel Tummala
Miami University
Ohio, USA
tummalvs@miamioh.edu

Daniela Inclezan
Miami University
Ohio, USA
inclezd@miamioh.edu

We introduce a framework for enabling policy-aware autonomous agents to reason about penalties for non-compliant behavior, and act accordingly. We use the $\mathcal{AOPL}$ language for policy specification and ASP for reasoning about policies and penalties. We build upon existing work by Harders and Inclezan on simulating the behavior of policy-aware autonomous agents and run tests on two different domains. We conclude that our framework produces higher quality plans than the previous approach.

1 Introduction

In this paper we explore autonomous agents that operate in dynamic environments governed by *policies* or *norms*. We introduce a framework that enables these policy-aware agents to reason about potential penalties for non-compliance and generate suitable plans for their goals. Unlike some of the previous work that only focused on compliant agents (e.g., [7, 1]), we believe that it is important to study the different nuances of non-compliant agent behavior for two reasons. First, autonomous agents may be tasked to accomplish high-stakes goals (e.g., assist in rescue operations) that may only be achievable if one or more non-compliant actions are executed as part of the plan. Second, this can be useful to policy makers if policy-aware agents are used to model human behavior since humans do not always comply to norms. In our proposed framework, policies are specified in the *Authorization and Obligation Policy Language* ($\mathcal{AOPL}$) by Gelfond and Lobo [3], and implemented in Answer Set Programming (ASP). We expand $\mathcal{AOPL}$ to enable the representation of, and reasoning about, penalties that may be incurred for non-compliance with a policy. Previous work on policy-aware autonomous agents [6, 4] defined various attitudes towards compliance with policies, called *behavior modes*. Non-compliant behavior was acceptable as part of the Risky behavior mode. However, in that framework the different plans containing non-compliant actions were ranked solely based on the length of the plan. Instead, our framework is capable of distinguishing between different non-compliant plans via penalties, so that an agent that disobeys certain policies to achieve its goal can do so with minimum repercussions. Based on our experiments, we conclude that our framework produces better quality plans. At least for some domains, our approach is also more efficient.

2 Background: Policy-Specification Language $\mathcal{AOPL}$

Gelfond and Lobo [3] designed the Authorization and Obligation Policy Language $\mathcal{AOPL}$ for specifying policies for an intelligent agent acting in a dynamic environment. A policy is a collection of authorization and obligation statements. An *authorization* indicates whether an agent's action is permitted or not, and under which conditions. An *obligation* describes whether an agent is obligated or not obligated to perform a specific action under certain conditions. An $\mathcal{AOPL}$ policy assumes that the agent's environment

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 108–110, doi:10.4204/EPTCS.416.8

is described in an action language such as $\mathcal{AL}_d$ [2]. An $\mathcal{AL}_d$ system description defines the domain's transition diagram whose states are complete and consistent sets of fluent literals and whose arcs are labeled by actions. The signature of an $\mathcal{AOPL}$ policy includes (a) the signature of the dynamic system description (which consists of predicates for *sorts*, *fluents*, and *actions*) and (b) predicates *permitted* for authorizations, *obl* for obligations, and *prefer* for specifying preferences between authorizations or obligations. An $\mathcal{AOPL}$ *policy* $\mathcal{P}$ is a finite collection of statements of the form:

$$
\begin{array}{rlll}
 & permitted(e) & \textbf{if } cond & (1a)\\
 & \neg permitted(e) & \textbf{if } cond & (1b)\\
 & obl(h) & \textbf{if } cond & (1c)\\
 & \neg obl(h) & \textbf{if } cond & (1d)\\
d : \textbf{normally} & permitted(e) & \textbf{if } cond & (1e)\\
d : \textbf{normally} & \neg permitted(e) & \textbf{if } cond & (1f)\\
d : \textbf{normally} & obl(h) & \textbf{if } cond & (1g)\\
d : \textbf{normally} & \neg obl(h) & \textbf{if } cond & (1h)\\
 & prefer(d_i, d_j) & & (1i)
\end{array}
$$

where e is an elementary action; h is a happening (i.e., an elementary action or its negation[1]); *cond* is a set of atoms of the signature, except for atoms containing the predicate *prefer*; d appearing in (1e)-(1h) denotes a defeasible rule label; and d_i, d_j in (1i) refer to distinct *defeasible* rule labels from $\mathcal{P}$. Rules (1a)-(1d) encode *strict* policy statements, while rules (1e)-(1h) encode *defeasible* statements (i.e., statements that may have exceptions). Rule (1i) captures *priorities* between defeasible statements only.

3 Penalization Framework for Policy-Aware Agents

We extend the $\mathcal{AOPL}$ syntax by a new type of statement for penalties:

$$penalty(r, p) \textbf{ if } cond_p$$

where r is the label of the prohibition or obligation rule for which the penalty is specified, p stands for the number of penalty points imposed if the rule r is broken, and $cond_p$ is a collection of static literals. As in Inclezan's work [5], we assume that all $\mathcal{AOPL}$ rules are labeled, including strict ones, which is not the case in the original definition of the language. The "**if** $cond_p$" part is omitted if $cond_p$ is empty. For instance, here is a strict policy rule labeled $r6(L)$ and its associated 3-point penalty. The rule says that the agent is obligated to stop at a location L whenever there are pedestrians crossing at that location:

$$
\begin{array}{l}
r6(L) : obl(stop(L)) \ \textbf{if} \ pedestrians\_are\_crossing(L)\\
penalty(r6(L), 3)
\end{array}
$$

Multiple penalty values can be associated with the same rule, based on different gravity levels. Let's consider the following defeasible policy rule $r1(L_1, L_2, S, S_1)$ saying that normally one is not permitted to exceed the speed limit by more than 5 mph if the speed limit is under 55 mph:

$$
\begin{array}{lll}
r1(L_1, L_2, S, S_1) : & \textbf{normally} & \neg permitted(drive(L_1, L_2, S))\\
 & \textbf{if} & speed\_limit(S_1),\ S > S_1 + 5,\ S_1 < 55
\end{array}
$$

The various levels of penalties for 1, 2, and 3 points respectively are assigned to the rule in $\mathcal{AOPL}$ as:

$$
\begin{array}{lll}
penalty(r1(L_1, L_2, S, S_1), 1) & \textbf{if} & S - S_1 < 10.\\
penalty(r1(L_1, L_2, S, S_1), 2) & \textbf{if} & S - S_1 >= 10,\ S - S1 < 20.\\
penalty(r1(L_1, L_2, S, S_1), 3) & \textbf{if} & S - S_1 >= 20.
\end{array}
$$

[1] If $obl(\neg e)$ is true, then the agent must not execute e.

Next, we develop ASP rules that can identify the penalties an agent incurs at each time step and specify which policy rule has been violated. We do so by introducing a predicate $add\_penalty(r, p, i)$, which says that a penalty of p points should be added to the total for breaking policy rule r at time step i. This predicate's definition addresses three scenarios: (1) when a prohibited action is included in the plan; (2) when a required action is missing from the plan; and (3) when a forbidden action is erroneously included in the plan.

$$add\_penalty(R,P,I) \leftarrow rule(R),\ holds(R,I),\ head(R,\neg permitted(E)),\ occurs(E,I),\ penalty(R,P)$$
$$add\_penalty(R,P,I) \leftarrow rule(R),\ holds(R,I),\ head(R,obl(E)),\ \neg occurs(E,I),\ penalty(R,P)$$
$$add\_penalty(R,P,I) \leftarrow rule(R),\ holds(R,I),\ head(R,obl(\neg e)),\ occurs(e,I),\ penalty(R,P)$$

where R is a rule, I is a time step, E is an elementary agent action, and P is a penalty.

The overall penalty for a plan is captured by the predicate $cumulative\_penalty$ defined using the $\#sum$ aggregate of CLINGO, and similarly for cumulative time to complete a plan:

$$cumulative\_penalty(N) \quad \leftarrow \quad \#sum\{P,R,I : add\_penalty(R,P,I)\} = N$$

An agent can either prioritize minimizing the cumulative penalty or minimizing the cumulative time when searching for an optimal plan.

4 Findings from Experiments

Our framework enhances the specification and simulation of agent behavior compared to the previous work by Harders and Inclezan (HI framework) [4], which lacked the concept of penalties. We tested our framework on the Rooms Domain [4] and observed significant improvements in time efficiency. When testing it on a newly developed Traffic Norms Domain, in which selecting an appropriate driving speed was fundamental, our framework produced higher quality plans. The HI framework chose random speeds, as it was penalty-agnostic. The choice of appropriate speeds resulted in longer execution times for our framework, as it required more optimization cycles compared to the HI framework.

References

[1] Natasha Alechina, Mehdi Dastani & Brian Logan (2012): *Programming norm-aware agents*. In: *Proceedings of AAMAS'12 - Volume 2*, Richland, SC, p. 1057–1064, doi:10.5555/2343776.2343848.

[2] Michael Gelfond & Daniela Inclezan (2013): *Some properties of system descriptions of ALd*. *J. Appl. Non Class. Logics* 23(1-2), pp. 105–120, doi:10.1080/11663081.2013.798954.

[3] Michael Gelfond & Jorge Lobo (2008): *Authorization and Obligation Policies in Dynamic Systems*. In Maria Garcia de la Banda & Enrico Pontelli, editors: *Logic Programming*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, pp. 22–36, doi:10.1007/978-3-540-89982-2_7.

[4] Charles Harders & Daniela Inclezan (2023): *Plan Selection Framework for Policy-Aware Autonomous Agents*. In Sarah Alice Gaggl, Maria Vanina Martinez & Magdalena Ortiz, editors: *Proceedings of JELIA'23*, *Lecture Notes in Computer Science* 14281, Springer, pp. 638–646, doi:10.1007/978-3-031-43619-2_43.

[5] Daniela Inclezan (2023): *An ASP Framework for the Refinement of Authorization and Obligation Policies*. *Theory and Practice of Logic Programming* 23(4), p. 832–847, doi:10.1017/S147106842300011X.

[6] John Meyer & Daniela Inclezan (2021): *APIA: An Architecture for Policy-Aware Intentional Agents*. In: *Proceedings of ICLP'21 (Tech. Comm.)*, *EPTCS* 345, pp. 84–98, doi:10.4204/EPTCS.345.23.

[7] Nir Oren, Wamberto Vasconcelos, Felipe Meneguzzi & Michael Luck (2011): *Acting on Norm Constrained Plans*. In João Leite, Paolo Torroni, Thomas Ågotnes, Guido Boella & Leon van der Torre, editors: *Computational Logic in Multi-Agent Systems*, Springer, pp. 347–363, doi:10.1007/978-3-642-22359-4_24.

Mind the Gaps: Logical English, Prolog, and Multi-agent Systems for Autonomous Vehicles

Galileo Sartor
Department of Computer Science, Swansea University, United Kingdom
galileo.sartor@swansea.ac.uk

Adam Wyner
Department of Computer Science, Swansea University, United Kingdom
a.z.wyner@swansea.ac.uk

Giuseppe Contissa
Department of Legal Studies, University of Bologna, Italy
giuseppe.contissa@unibo.it

In this paper, we present a modular system for representing and reasoning with legal aspects of traffic rules for autonomous vehicles. We focus on a subset of the United Kingdom's Highway Code (HC) related to junctions. As human drivers and automated vehicles (AVs) will interact on the roads, especially in urban environments, we claim that an accessible, unitary, high-level computational model should exist and be applicable to both users. Autonomous vehicles introduce a shift in liability that should not bring disadvantages or increased burden on human drivers. We develop a system "in silico" of the model. The proposed system is built of three main components: a natural language interface, using Logical English, which encodes the rules; an internal representation of the rules in Prolog; and an multi-agent-based simulation environment, built in NetLogo. The three components interact: Logical English is translated into and out of Prolog (along with some support code); Prolog and NetLogo interface via predicates. Such a modular approach enables the different components to carry different "burdens" in the overall system; it also allows swapping of modules. Given NetLogo, we can visualize the effect of the modeled rules as well as validate the system with a simple dynamic running scenario. Designated agents monitor the behaviour of the vehicles for compliance and record potential violations where they occur. The information on potential violations is then utilized by Validators, to determine whether the violation is punishable, differentiating between exceptions and cases.

1 Introduction

Research in autonomous vehicles is improving at a continuous pace. In a possible future, AVs and human agents will share a common space, such as city streets, where the AVs will have to interact with other road users in a predictible and understandable way.

Among the problems that need to be addressed specifically for this shared scenario is that of making the behaviour of AVs conform to the traffic laws [14], in such a way as to not increase the burden on the other users of the roads.

For this reason we propose and present the development of rules from the UK Highway Code (traffic rules) that are modelled for both human and autonomous drivers.

The hypothesis we put forward is that with a combination of logic modelling and natural language we can obtain a representation of norms that can be directly used by both humans and autonomous agents, thus simplifying the inclusion of AVs on mixed roads.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 111–124, doi:10.4204/EPTCS.416.9

The system we present does not make use of machine learning (e.g., image recognition) as input to the rule base. Rather we assume that the readings of the vehicle sensors are correct and fed into the rule base, so that the vehicle can reason on the input and determine which action to take accordingly. In future work, we might consider an integration between a rule-base and machine learning.

The system is designed to simulate multiple agents on the road, both human and autonomous and to detect violations in their behaviour. The violations are recorded by specific agents (monitors). The logs, with the information coming from the offending vehicle, can be used in a second moment to perform legal reasoning on the violation and assign penalties when necessary.

The paper presents the background, outlining the main goals and issues in 2, with particular attention on the issue of liability in 3 and the existing projects/papers that explore the same or a similar space outlined in 4. The modular structure of the proposed system is then presented in 5, followed by the methodology for the design of the different component in 6, with examples of the code. The legal issues analyzed in the paper are further explored in the context of the proposed system in 7, specifically looking at how such a system could reason on the occurring violations, keeping in mind what the monitors logged, and also the reasoning the vehicles made at the time of the violation. This section will also look at what it means to violate a rule in the context of driving, what types of violations could occur, and their differing legal outcomes. We summarise and discuss future work in Section 8.

2 Background

Autonomous vehicles are going to share the road with human drivers, and other non autonomous agents. It is therefore crucial to ensure the behaviour of AVs is consistent with that of a *good driver*, i.e. a human driver who follows the rules in a predictable manner.

This is even more important if we consider that, if well developed, these vehicles could be aware of their decision making, and could provide understandable explanations of the reasoning behind a certain action. As we will cover in the following sections, we consider this to be a crucial point, to reduce the burden on human agents who interact with the vehicle, and to reason with violations and reparations.

In developing the system, we should also consider the fact that as humans we make decisions also based on the possibility of incurring in violations and fines. There may be multiple reasons for this, and some may be legally valid, such as in the case of rules with exceptions, as will be described later. In the case of AVs however the general behavioural rules have to be determined at build time, given that there is no human agent involved in taking decision while driving. This question is made more complicated by the issue of liability.

3 Liability

While autonomous vehicles are expected to drive more consistently with respect to the law and reduce accidents, violations of the law and accidents may still happen in certain situations. The question that arises is who should be held responsible for such violations and accidents - the driver, manufacturer, or the algorithm developer? Such issues are relevant for the general context of development of autonomous vehicles and guide how a system might be modeled. We develop issues below and specifically tie the concept of “lawful reasonable agent” to our implementation.

At the highest levels of automation, that is levels 4 and 5, according to the SAE definition [16], the autonomous vehicle takes full control of all dynamic driving tasks. In these two levels of automation, the user is not expected to intervene when the automated driving system is on. Therefore, user’s liability is to be considered only when the ODD limits are exceeded (namely, the limits within which the driving automation system is designed to operate), or when users request the system to disengage the automated driving system. [1]

It should be noted that vehicles under level 4 or 5 may even be designed to be exclusively operated by technology, that is without user interfaces such as braking, accelerating, steering, etc. Therefore, such categories of vehicles do not contemplate at all the role of the human driver. Whenever user interfaces and controls are missing, the user will not be able to intervene in any of the dynamic driving tasks, and therefore cannot be considered in any way responsible for driving and subject to the related liabilities [7].

Therefore, under levels 4 and 5, the burden of liability is mostly on manufacturers and/or programmers. They would be liable (1) when providing a defective or non-standard compliant tool that had a role in the causation of the accident, and (2) whenever the system fails to carry out the assigned task with a level of performance that is (at least) comparable to that reached by a human adopting due care under the same conditions.

This is linked to the idea that there would be a reasonable expectation that the autonomous vehicle performs an assigned task in a way that ensures the same level of safety that would be expected from a human performing the same task. Thus, it would seem appropriate to compare the autonomous vehicle's behaviour in carrying out an assigned task, with the behaviour that would be otherwise expected by the human driver [13].

In this perspective, the concept of negligence would be central in evaluating the behaviour of the autonomous vehicle and assessing liabilities for manufacturers and programmers.

Negligence is 'the omission to do something which a reasonable man, guided upon those considerations which ordinarily regulate human affairs, would do, or doing something which a prudent and reasonable man would not do' (Blythe v Birmingham Waterworks (1856) 11 Exch 781, at p 784).

The tort of negligence usually requires the following elements: the existence of an injury, harm or damage; that the injurer owes *a duty of care* to the victim; that the injurer has broken this duty of care (fault); that the damage (or injury) is a reasonably relevant consequence of the injurer's carelessness (causal connection)[5]. In the legal discourse, "negligence" denotes carelessness, neglect, or inattention, which are mental stances that can be ascribed only to human minds.

A driver of the vehicle has an asymmetrical duty of care toward pedestrians or other individuals in the vicinity. The rules of 170-183 in part characterise how to meet this duty of care; broadly speaking, the driver should proceed defensively and cautiously, anticipating behaviours of others which might create circumstances in which the likelihood of an accident increases.

Clearly, the concept of negligence is linked to the idea of a human fault. In contrast, liability for technological failures is usually evaluated and allocated on the grounds of product liability, which requires evidence of the following: an injury, harm or damage; a defective technology; and a causal connection between the damage (or injury) and the defect, namely that the former must be a reasonably relevant consequence of the latter. A technology may be considered defective if there is evidence of a design defect, a manufacturing defect, or a warning defect. Design defect, where the design is unreasonably unsafe, is the most relevant, and is usually determined by courts taking into account one of the following tests: the state of the art, the evidence of alternative design, or the reasonable expectations of users/consumers with regard to the function of the technology[17]. Key for our purposes is that negligence and product

[1] This is consistent with the 2022 update to the Highway code, stating that *While a self-driving vehicle is driving itself in a valid situation, you are not responsible for how it drives. You may turn your attention away from the road and you may also view content through the vehicle's built-in infotainment apparatus, if available.*

liability contrast with respect to duty-of-care and defective technology.

Yet, human and AI liability converge around performance: when a technology is used in substitution of a human, there is a reasonable expectation that the AI will perform an assigned task in a way that ensures the same level of safety that would be expected from a human performing the same task.

We reasonably assume that technology is presumed to have at least the same level of performance and safety as the human user. Thus, an AV ought to bear the same liability and duty of care as a human user. We propose that one code of conduct should rule both human and autonomous driver as a matter of fairness and equality on the road. This implies that we ought to be able to interrogate the automated vehicle on the same grounds as the human driver. To realise this, our modeling language should provide a unitary model, which yet allows for alternative means of realisation and data input.

4 State of the Art

In AI and Law, one of the main goals is to represent legal provisions as code [8, 1]. There are issues to address in order to obtain a good representation faithful to the source, and the main one is the presence of vagueness and open texture in the law [2]. [4] discusses the legal context related to open textured concepts and defeasibility. The natural language version of the HC has similar issues, that have already been discussed in the context of AVs with reference to natural language [10] and commonsense reasoning [11].

There have been multiple proposed approaches to addressing the issue of autonomous vehicles and the rules of the road.

An possible issue with the purely data-driven approaches is that there is a lack of well formed, diverse datasets [9], that are often biased towards accidents. Furthermore, they rely heavily on the interaction between autonomous vehicles [18], that are able to communicate, and act together as a swarm.

In this research we are attempting to address the interaction with and expectations of human agents on shared roads, where vehicles cannot rely on inter-vehicle communication. Further research is ongoing on how to apply data-driven predictive systems to mixed traffic (human and AVs).

We will focus on the rule-based approaches, and in particular those that enable further legal reasoning on the occurring events and actions.

In the research considered, the rules of the road (or a subset) have been modelled in higher order logic or temporal logic, to reason about the desired actions and concepts about the environment, in order to determine whether a certain action is valid. The agent in question can then take the desired action, and proceed with the movement.

One example of a similar representation is that of the RoTRA (Rules of The Road Advisor), [6], in which the rules are encoded in Prolog, and queried with respect to the state of the world (the context and beliefs), and the desired goal (intention).

Other projects have a more narrow focus on specific issues, such as determining the safe overtaking distances, with a formal model developed in Linear Temporal Logic, implemented in Isabelle/HOL, [15].

The issue of encoding and reasoning with commonsense knowledge is not specific to the domain of autonomous vehicles, and is in fact a broader issue of knowledge based systems. In the context of driving, the analogy with human reasoning, and how modelling commonsense reasoning can help to develop reliable autonomous vehicles, is the topic of the AUTO-DISCERN (AUTOnomous DrivIng uSing CommonsEnse ReasoniNg) project[11].

[3] presents an automatic compliance checking framework to assess AVs behaviour with respect to the traffic rules of Queensland, Australia. It considers issues related to open texture, exceptions, and

conflicts amongst rules.

In our research, we assume the driving environment has not been sterilised of human drivers, but rather includes them. Human understanding of and behaviour in the driving environment must be taken into consideration, which may go beyond the specification of the traffic rules (i.e., the interpretations) and require a unitary, transparent representation for both sorts of drivers, ensuring consistency. We are also interested in the legal reasoning that occurs on the detected potential violations, and how the information from the vehicles can be used to reason about the specific scenario.

5 Structure

The system presented is split in different, mostly independent modules, that each deal with one of the requirements and interact through minimal translation layers.[2]

The controlled natural language (CNL) module is written in Logical English [12], syntactic sugar on Prolog, that enables to write rules and interact with the system in natural language. Using Logical English we can represent logic rules in natural language, that can be directly queried with a Prolog interpreter, or translated to Prolog for use by the autonomous vehicle.

The Logic rules module is written in Prolog, and is mostly derived from the Logical English representation. The autonomous vehicle can reason with a Prolog interpreter, and use the result in determining its driving behaviour. The Prolog output can be logged or converted back in natural language, saved in a human readable format, and can be used to check instances after the fact (scenarios and queries). This could be used in case of accidents or violations to determine why the vehicle took certain actions.

The simulation module uses NetLogo, a multi-agent programmable modeling environment, where vehicles with different properties are spawned, ad can move around on a predefined road grid. In addition to the basic movement, the vehicles can query the LE/Prolog rulebase to determine whether they are allowed to perform a certain action, or conversely, if they are prohibited.

In the following section the division of labour between the different components that was chosen for the system will be made clear.

6 Methodology

The development of the system started with the representation of norms in Logical English [Cite Mind the Gap]. Given the need for a simulation system, and the availability of different potential candidates, the idea was to keep the system modular, with the possibility to swap different components. The current simulation uses NetLogo, but there is limited overlap in the components, mainly what is needed to convert data and I/O. The rules themselves can still be queried by LE/Prolog, and combined with other models. The NetLogo simulation is derived from one of the examples made available in NetLogo[3], and is then expanded through the use of a bridge to Prolog[4] that had previously been developed, and has been updated for the purpose of this project. Vehicles in NetLogo are assigned different properties, and

[2] The full code of the system is available at `https://github.com/LyzardKing/mind-the-gap/tree/ICLP2024`

[3] `https://ccl.northwestern.edu/netlogo/models/TrafficIntersection`

[4] `https://github.com/LyzardKing/NetProLogo`

are spawned at one of the edges of the screen. They follow the road, and decide whether to turn at an intersection randomly. At the moment the system is not responsible for route planning, although in the future it might be added. There are multiple types of vehicles, each with particular properties. Some of the properties are reflected in Prolog, while others are limited to the NetLogo environment. Most properties and data come from sensors in the vehicle, that perceive the surrounding environment, as well as the properties inherent to the vehicle itself. The vehicles can see their surroundings, avoid other objects/agents, without accessing the legal norms. If we disable the Prolog section the vehicles can still drive, and act more like a swarm (Cite). This behaviour may be very efficient on roads only used by autonomous vehicles, where the vehicles can communicate, and organize their actions accordingly. This is not the case on mixed roads, with both human and autonomous agents, pedestrians, bikes, emergency vehicles, and other potentially unpredictable agents. Human drivers, while sharing the road with AVs, will have to be able to trust and understand the actions of the surrounding vehicles, while not necessarily knowing (or caring) if they are human or autonomous. In these circumstances, AVs will have to respect the same rules as their human counterparts, even if the actions are less optimized.

6.1 Logic rules

In the system the rules are represented in Logical English, in a way that is as isomorphic as possible to the original text. This makes the rules readable by humans, and could point to the possibility of having one simple corpus on which to write the rules, with them being automatically understandable and implemented by humans and autonomous agents. The main goal here is to avoid repeating and maintaining multiple codebases, and to ensure the logic structure of the natural language version of the rules. To assess the viability of such an approach the first rules modelled were those dealing with junctions (Rules 170-183 of the Using the Road section of the HC). The first thing to note is that we are dealing with different types of morns, that may have different consideration when modelling: rules with a highlighted MUST, or MUST NOT, are those that are tied directly to laws (the Road Traffic Act 1988, The Traffic Signs Regulations and General Directions 2002), and deal with cases in which the driver is considered is guilty of an offence. We will visit these cases more in the next sections. Most other rules deal not with explicit prohibitions/obligation, rather dictate what the behaviour of a good driver should be. In this case the terminology of the HC is very different, using words such as should, take extra care, look around, … These terms are more nuanced, and while as humans we know how to deal with them, the same cannot be assumed of autonomous agents. For the representation to be adequate enough we may need in certain cases to add more information, and additional rules that form part of our commonsense reasoning. Let us consider one of the modelled rules, rule 171, which states that:

> You MUST stop behind the line at a junction with a 'Stop' sign and a solid white line across the road. Wait for a safe gap in the traffic before you move off.

This rule could be modelled by identifying the goal of the vehicle, entering the junction, and building the rule in Listing 1. In the Logical English code, the word *can* means *has the permission to*, as used in the Highway Code.

In this case the rule expresses what should happen to the vehicle when approaching the junction. At first the vehicle should stop, since it is approaching a stop sign. Once it is next to the stop sign, the vehicle can query the system for its permission to enter the junction, and the second rule would be evaluated. This is how the rules are currently modelled, and through further revisions they could be made more isomorphic depending on the specific needs. The rules can be queried as is, by giving a scenario, a sample query that an AV could make, and could consequently show the solution in natural language, with

```
a vehicle can enter the junction if
the vehicle is of type ambulance
and it is not the case that
    the vehicle must give way to an other vehicle.

a vehicle can enter the junction if
the vehicle has green light
and it is not the case that
    the vehicle must give way to an other vehicle.
```

Listing 1: “Example rule in Logical English”

all the steps that have been used to derive a certain conclusion (trace). We are interested however in a more dynamic use of the rulebase, and for that we can rely on the previously mentioned agent simulation.

6.2 Agent simulation

The simulation is running currently in NetLogo, with the prolog extension to enable the agents to query the rulebase. While currently there is only one Prolog process running, the single queries made by the vehicles are independent and isolated, to ensure that the queries are all atomic, and simulate a realistic scenario.

The rules in NetLogo are only those that pertain to the physical constraints, e.d. those actions the vehicles cannot physically make (e.g., occupying the same space of another vehicle). It is thus possible for the vehicles to drive without additional rules (in the same way as it is possible for human drivers to ignore the rules of the road). We then introduce the “legal” constraints, the Prolog rules.

With the addition of the rules from the HC, the behaviour of the AVs becomes closer to what we would expect from human drivers.

Environment The simulated environment is very simple, consisting of three roads with two intersections. one of the intersections has a traffic light, while the other has stop signs. The intersections are spawned with their specific properties, and agents are generated independently starting from random road sections.

6.2.1 Agents

In the simulation there are different agents, with different properties and goals:

Vehicles Vehicles are divided in two categories: cars, and emergency vehicles (ambulances). This is because rules may apply differently to different vehicle types. At the moment in the simulation vehicles are divided in two categories: autonomous and human; as well as two types: cars and ambulances.

The different types of vehicles can be expanded, and share certain rules. In particular the main difference between the ambulances and cars is that the ambulances can violate certain rules of the HC, like crossing with a red light, so long as that doesn’t directly cause an accident. To check this the vehicle uses the information about its surroundings to determine whether there is another vehicle close enough.

The cars are split in human and machine driven, with the main difference for now is the introduction of a number of delays and variations in the human behaviour. For example, a human driver may decide to go faster than the speed limit.

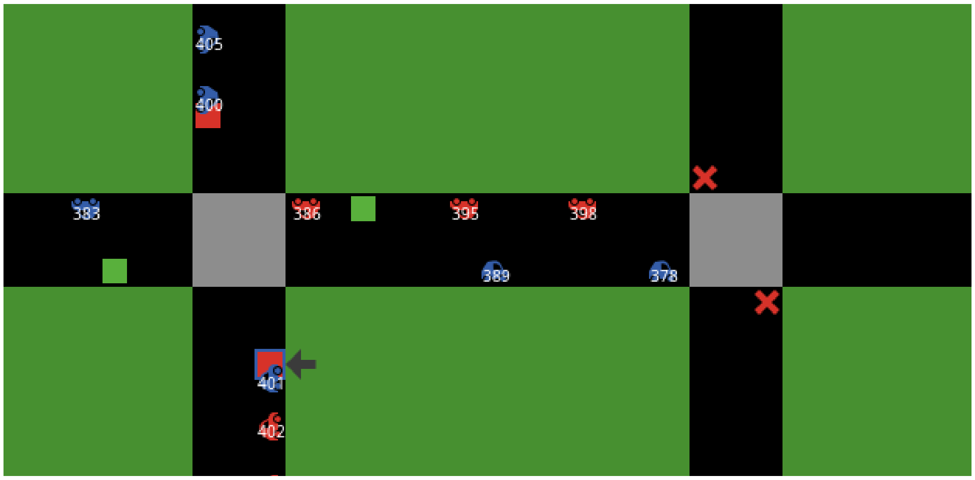


Figure 1: Simulated environment

```
a vehicle potentially violates entering the junction if
the vehicle has red light
and it is not the case that
    the vehicle is stopped.
```

Listing 2: “Example of the modeled violations”

Pedestrians In the current simulation the pedestrian have a very basic behaviour, simply crossing when they get to a road. The only thing pedestrians will look at is if there is a car immediately approaching. This behaviour will be expanded upon in future development.

Monitors Monitors are the final agents that are active in the simulation. They focus on one section of road to see if they detect any vehicles which may have violated a traffic rule. The monitors have a narrow scope of vision and only access visible properties in the environment, e.g., cameras that recognize the license plate, speed and position of the vehicles; they only react with respect to that information. As such, monitors are purely reactive, rather than interactive. Moreover, they do not do any legal reasoning per se, which is why we only say they identify whether a vehicle may have violated a traffic rule a vehicle with its scope of coverage.

The rules that pertain to the monitors are modelled as in Listing 2. In the simulation, the monitor has vision of the traffic light, the vehicle position, and speed. When a vehicle enters the cone of vision of the monitor, the monitor gathers information about the traffic light and the vehicle speed; the monitor can detect whether the vehicle is moving or not, passing the predicate “vehicle is stopped” and the traffic light colour to Prolog rules. The Prolog rules used by the monitors then determine if a *potential violation* occurred, i.e., if the vehicle is not stopped and the light is red.

As discussed later, information on potential violations is passed to a *validator*, which may have additional information about the properties of a vehicle which can be taken (or not) to mitigate against issuance of a reparation. We say that if there is a potential violation and no mitigating circumstances, then there is a *punishable violation*, which leads to a reparation.

7 Violations and Penalties

While the HC is not in itself formally a legally binding document, it contains legal rules and references to the law, which indicate when violations arise and what is the correlated reparation (i.e., penalty to be paid; for our purposes, we use reparation and penalty interchangeably).

For instance: Failing to comply with traffic sign; Road Traffic Act 1988, s.36; £100; points 3. Also see Road Traffic Offences Act 1988 `https://www.legislation.gov.uk/ukpga/1988/53/part/II`. And the schedules with the penalties, for example, `https://www.legislation.gov.uk/ukpga/1988/53/schedule/2/part/I`. Our simulation must act and reason with respect to the violations and reparations.

In the simulation, vehicles can violate the HC rules in certain situations. A human driver can independently decide if it is worth breaking a rule, depending on many factors, such as the probability of being caught, the probability of accidents, the change in time to reach the destination, and the amount of the potential fine. Furthermore, whether or not a penalty is applied to an instance of a violation might depend on whether it is "excusable' for one reason or another; that is, a violation is an exception to a norm, but some violations can themselves be exempted from penalty. For instance, a driver might be caught speeding, but not pay a penalty as they explain they were handling a medical emergency. This general list of factors mostly applies to AVs as well with a caveat that there is no driver responsible (and liable) for deciding to break a rule, nor for the possible consequences. As can be imagined, there are many factors with respect to which a norm is violated and conditions under which a penalty is or is not applied.

Given this, we work with a small domain to implement a vehicle's actions executed with respect to rules, whether the vehicle's action violates the rule, the detection of violations, consideration of mitigating circumstances, and the consequential penalties. As there are several rules, each related to actions; there can be correlated distinct violations, detections, mitigating circumstances, and penalties. In a sense, then, the *actual* behaviour of a vehicle with respect to the rules of the road is compared to and evaluated against the *ideal* behaviour as specified by the rules of the road. The *ideal* behaviour is what the *lawful reasonable agent* of Section 3 would strive to achieve. Deviations of actual from ideal are noted and reasoned with further in terms of whether there were mitigating circumstances or not.

7.1 Design

Figure 2 is a graphic outline of the flow of information and reasoning. We start with Vehicle Scenario which is the state of the world within the scope of vision of the vehicle; it is the context in which the vehicle would execute an action (Vehicle Action). The Monitor is a reactive agent which is in charge of detecting a violation within its scope of vision which is the Monitor Scenario; they stand-in for cameras or the police. As a reactive agent, they record a Potential Violation, which remains to be validated with respect to the laws as indicated below. The Validator Scenario is a hypothetical state of the world, one in which the Vehicle Scenario has been modified were the goal of the Vehicle Action to be attained. The Validator Scenario is used by the Validator to scope consideration of the Lawful Actions, which are those actions which are compliant with the laws in that Validator Scenario; in effect, we are given all those actions which, were they executed in the given Validator Scenario would be lawful. The Validator is triggered by an instance of a Potential Violation; it is used to evaluate whether the Potential Violation is indeed illegal or whether there might be mitigating circumstances. To move to this next step (VA in LA wrt PV), we consider whether the action that the vehicle executes (Vehicle Action) is amongst the Lawful Actions relative to the relevant Potential Violation, that is, whether the action has been caught by

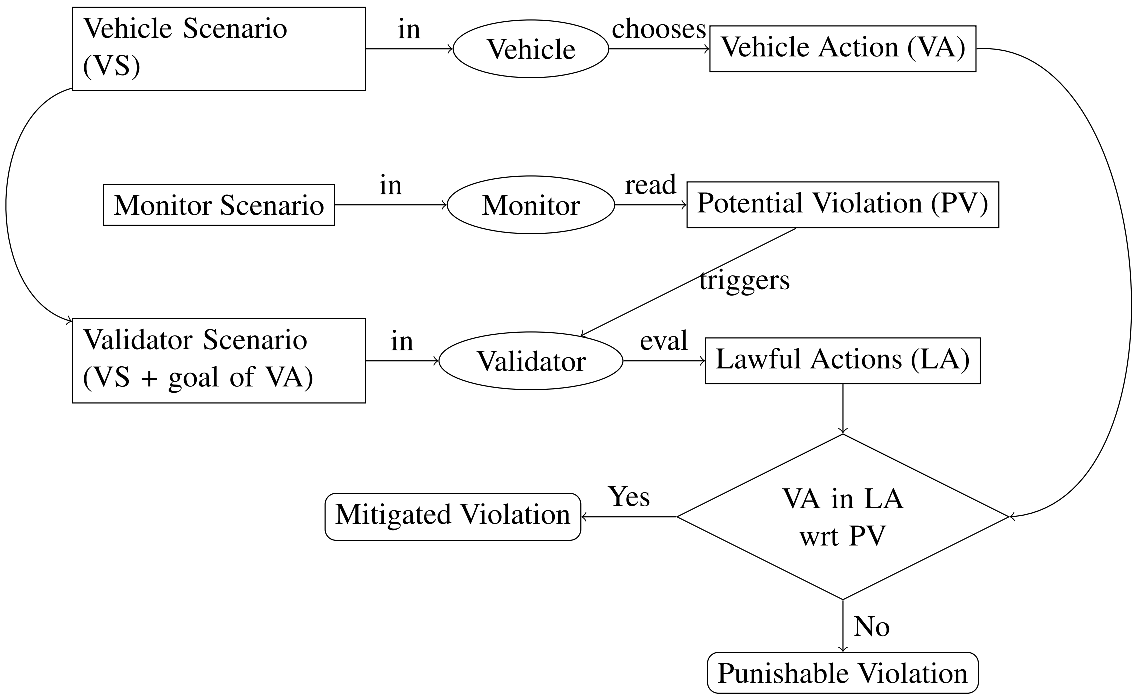


Figure 2: Action Execution with respect to Legal Rules

a monitor for a possible legal violation. If it is, then the violation is a Mitigated Violation; if not, then it is a Punishable Violation, which could require a penalty payment.

For example, in Figures 3 and 4, we have vehicles which enter the intersection against the red light. This introduces a Potential Violation in that whether the vehicle is penalised depends on whether or not it has mitigating circumstances relative to the law. An ordinary vehicle would raise a Punishable Violation in Figure 3, from which we would infer a correlated reparation (not shown). However, an ambulance would raise a Mitigated Violation, as in Figure 4, since as an ambulance it has a legitimate reason not to abide by the rule; consequently, no reparation can be inferred.

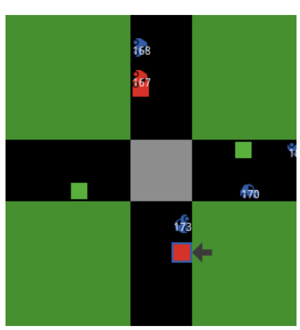


Figure 3: Example of violation detected by the monitor (the arrow)

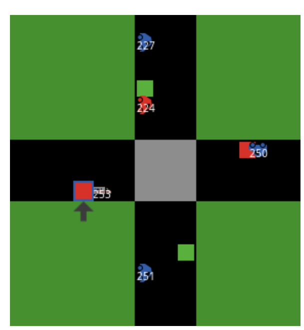


Figure 4: Example of an "allowed" violation, i.e., a permission

7.2 Implementation

Here we outline the implementation for each component of the design.

Scenario and Vehicle Action Two possible Scenarios are in Listing 3 and 4, which contain the goal of entering the junction. The vehicle would execute an action (Vehicle Action), applying the rules in Listing 1 to the Scenario, which provides rules for each of a car and an ambulance. Note that an ambulance does not need to abide by red lights, while a normal vehicle does.

```
scenario:
    173 is of type car.
    173 has red light.
goal:
    173 can enter the junction
    .
```

Listing 3: "Normal vehicle Scenario"

```
scenario:
    253 is of type ambulance.
    253 has red light.
goal:
    253 can enter the junction
    .
```

Listing 4: "Ambulance Scenario"

Monitor The monitor can detect a violation, as discussed in relation to Listing 2, and records it.

Validator Scenario is a hypothetical state of the world, one in which the Vehicle Scenario has possibly been modified where the goal would be realised by the Vehicle Action. In this instance, since the Potential Violation is related to the goal that the vehicles had in the Scenario and Vehicle Action above, the Validator Scenario is equivalent to the Vehicle Scenario. These need not be equivalent; for example, if the vehicle were caught speeding, though its goal were entering the junction.

Validator The Validator uses the rules in Listing 5 with the Rules of the Road of Listing 1 to determine the possible lawful actions, and compare them to the action which gives rise to the Potential Violation.

Comparing the Vehicle Actions, Legal Actions, and Potential Violations The Listing 5 uses information about the recorded Potential Violation and whether the vehicle can execute the action (Listing 1). Where the vehicle can cross the red light and it is an ambulance, there are mitigating circumstances, so a violation is mitigated; where the vehicle cannot cross the red light, the violation is punishable.

```
a vehicle punishably violates an action if
the vehicle potentially violates the action
and it is not the case that
    the vehicle can the action.

a vehicle mitigately violates an action if
the vehicle potentially violates the action
and the vehicle can the action.
```

Listing 5: "Comparing Ideal (Legal) Actions to Real (Violable) Actions"

8 Summary and Future Work

We have presented a modular framework for modeling autonomous vehicles that respect or violate the rules of the road and interact with other road users. The modeled vehicles are designed in a way that makes their behaviour compatible with the behaviour of human agents, particularly with respect to violability. The model includes violation detection and evaluation in a way that can take into account some different cases and exceptions. As a modular system, components can be replaced with others, making it more complex, while maintaining the basic legal considerations and provisions. The overall function as outlined in Figure 2, while the specific examples present a simplified instance.

In future work, we intend to report other aspects of the implementation and continue this research, expanding the rule-base and the simulation to closer map real world scenarios. The legal reasoning component will be expanded, analyzing different natural language representations of rule priorities, exceptions, and the respective logic formulations.

The goal is to keep the rules modeled in a CNL as close as possible to the original source text. This may require tweaking parts of the existing code to ensure it is compliant with this requirement. In particular the definition of exceptions and rule hierarchy should be easily understandable and intuitive to human readers.

A possible line of research deals with the (partially) automated extraction of rules from the original source, so that it can be modeled as logic by an automated system and subsequently verified by human experts. This process would involve validating existing tools to automate the parsing of the text, and in particular their ability to keep the necessary level of consistency between the different rules.

Integration of machine learning approaches with legal reasoning would be an important avenue to explore, though how and where it integrates is an open question. While we would want to "hard code" from the HC, the overall system should have some flexibility to account for a variety of circumstances, e.g., open texture and commonsense reasoning.

The analysis of the violations, and the legal reasoning that occurs after the violations have been detected, will be expanded, to better define the rules that apply to AVs, and how the AV could behave in situations where a human driver would perhaps decide to violate a rule. As part of this, some integration with planning would be essential.

Acknowledgements

The authors wish to thank Prof. Bob Kowalski, for his active leadership of the Logical English project, and for his encouragement to explore its applications to many different domains. We also thank Prof. Giovanni Sartor for supporting this work in the context of the H2020 ERC Project "CompuLaw" (G.A. 833647).

References

[1] Trevor Bench-Capon, Michał Araszkiewicz, Kevin Ashley, Katie Atkinson, Floris Bex, Filipe Borges, Daniele Bourcier, Paul Bourgine, Jack G. Conrad, Enrico Francesconi, Thomas F. Gordon, Guido Governatori, Jochen L. Leidner, David D. Lewis, Ronald P. Loui, L. Thorne McCarty, Henry Prakken, Frank Schilder, Erich Schweighofer, Paul Thompson, Alex Tyrrell, Bart Verheij, Douglas N. Walton & Adam Z. Wyner (2012): *A history of AI and Law in 50 papers: 25 years of the international conference on AI and Law*. *Artificial Intelligence and Law* 20(3), pp. 215–319, doi:10.1007/s10506-012-9131-x.

[2] Trevor Bench-Capon & Pepijn R. S. Visser (1997): *Open Texture and Ontologies in Legal Information Systems*. In Roland R. Wagner, editor: *Eighth International Workshop on Database and Expert Systems Applications, DEXA*, IEEE Computer Society, pp. 192–197, doi:10.1109/DEXA.1997.617268.

[3] Hanif Bhuiyan, Guido Governatori, Andy Bond & Andry Rakotonirainy (2024): *Traffic rules compliance checking of automated vehicle maneuvers*. *Artif. Intell. Law* 32(1), pp. 1–56, doi:10.1007/S10506-022-09340-9.

[4] Brian H. Bix (2012): *Defeasibility and Open Texture*. In Jordi Ferrer Beltrán & Giovanni Battista Ratti, editors: *The Logic of Legal Requirements: Essays on Defeasibility*, Oxford University Press, doi:10.1093/acprof:oso/9780199661640.003.0011.

[5] Gert Brüggemeier (2006): *Common principles of tort law: a pre-statement of law*. British Institute of International and Comparative Law.

[6] Joe Collenette, Louise A. Dennis & Michael Fisher (2022): *Advising Autonomous Cars about the Rules of the Road*. In Matt Luckcuck & Marie Farrell, editors: *Proceedings Fourth International Workshop on Formal Methods for Autonomous Systems (FMAS)*, 371, pp. 62–76, doi:10.4204/EPTCS.371.5.

[7] Giuseppe Contissa, Francesca Lagioia & Giovanni Sartor (2018): *Liability and automation: legal issues in autonomous cars*. *Network Industries Quarterly* 20(2), pp. 21–26.

[8] Guido Governatori, Trevor Bench-Capon, Bart Verheij, Michal Araszkiewicz, Enrico Fran-cesconi & Matthias Grabmair (2022): *Thirty Years of Artificial Intelligence and Law: The First Decade*. *Artificial Intelligence and Law* 30(4), pp. 481–519, doi:10.1007/s10506-022-09329-4.

[9] Ahmad Hammoud, Azzam Mourad, Hadi Otrok & Zbigniew Dziong (2022): *Data-Driven Federated Autonomous Driving*. In Irfan Awan, Muhammad Younas & Aneta Poniszewska-Marańda, editors: *Mobile Web and Intelligent Information Systems*, Lecture Notes in Computer Science, Springer International Publishing, Cham, pp. 79–90, doi:10.1007/978-3-031-14391-5.

[10] Patrick Irvine, Antonio A. Bruto Da Costa, Xizhe Zhang, Siddartha Khastgir & Paul Jennings (2023): *Structured Natural Language for Expressing Rules of the Road for Automated Driving Systems*. In: *2023 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1–8, doi:10.1109/IV55152.2023.10186664. Available at `https://ieeexplore.ieee.org/document/10186664`.

[11] Suraj Kothawade, Vinaya Khandelwal, Kinjal Basu, Huaduo Wang & Gopal Gupta (2021): *AUTO-DISCERN: Autonomous Driving Using Common Sense Reasoning*. In Joaquin Arias, Fabio Aurelio D'Asaro, Abeer Dyoub, Gopal Gupta, Markus Hecher, Emily LeBlanc, Rafael Peñaloza, Elmer Salazar, Ari Saptawijaya, Felix Weitkämper & Jessica Zangari, editors: *International Conference on Logic Programming 2021 Workshops*, *CEUR Workshop Proceedings* 2970, CEUR. Available at `https://ceur-ws.org/Vol-2970/#gdepaper7`.

[12] Robert Kowalski, Jacinto Dávila & Miguel Calejo (2021): *Logical English for legal applications*. *Conference: XAIF, Virtual Workshop on XAI in Finance*.

[13] Francesco Paolo Patti et al. (2019): *Autonomous vehicles' liability: need for change?* In: *Digital revolution-new challenges for law: data protection, artificial intelligence, smart products, blockchain technology and virtual currencies*, CH Beck, pp. 190–213.

[14] Henry Prakken (2017): *On the problem of making autonomous vehicles conform to traffic law*. *Artif. Intell. Law* 25(3), pp. 341–363, doi:10.1007/s10506-017-9210-0.

[15] Albert Rizaldi, Jonas Keinholz, Monika Huber, Jochen Feldle, Fabian Immler, Matthias Althoff, Eric Hilgendorf & Tobias Nipkow (2017): *Formalising and Monitoring Traffic Rules for Autonomous Vehicles in Isabelle/HOL*. In Nadia Polikarpova & Steve Schneider, editors: *Integrated Formal Methods*, Cham, pp. 50–66, doi:10.1007/978-3-319-66845-1.

[16] SAE On-Road Automated Vehicle Standards Committee (2021): *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*. SAE international.

[17] Hanna Schebesta (2017): *Risk Regulation Through Liability Allocation: Transnational Product Liability and the Role of Certification*. *Air and Space Law* 42(2), pp. 107–136, doi:10.54648/AILA2017011.

[18] Jiawei Wang, Yang Zheng, Qing Xu & Keqiang Li (2022): *Data-Driven Predictive Control for Connected and Autonomous Vehicles in Mixed Traffic*. In: *2022 American Control Conference (ACC)*, pp. 4739–4745, doi:10.23919/ACC53348.2022.9867378.

Simulating Supply-Chain Contract Execution: A Multi-Agent Approach (Extended Abstract)

Long Tran Tran Cao Son Dylan Flynn Marcello Balduccini
New Mexico State University Saint Joseph's University
longtran@nmsu.edu stran@nmsu.edu df752850@sju.edu mbalducc@sju.edu

Supply chains exhibit complex dynamics and intricate dependencies among their components, whose understanding is crucial for addressing the challenges highlighted by recent global disruptions. This paper presents a novel multi-agent system designed to simulate supply chains, linking reasoning about dynamic domains and multi-agent systems to reasoning about the high-level primitives of the NIST CPS Framework. Our approach synthesizes existing research on supply chain formalization and integrates these insights with multi-agent techniques, employing a declarative approach to model interactions and dependencies. The simulation framework models a set of autonomous agents within a partially observable environment, and whose interactions are dictated by contracts. The system dynamically reconciles agents' actions, assessing their feasibility and consequences. Based on the state of the domain, the simulation framework also draws conclusions about the high-level notions of requirements and concerns of the NIST CPS Framework, which provide a uniform and domain-agnostic vocabulary for the understanding of such complex systems as supply chains.

1 Introduction

Building on our previous work on formalizing supply chains and the underlying contracts [2], this paper introduces a simulation of the dynamics of supply that relies on a view of the supply chain as a multi-agent system. Central to our approach is the use of formal contracts, which define the obligations and interactions between agents within the simulation environment. Our simulation integrates an additional knowledge layer based on the National Institute of Standards and Technology's Cyber-Physical Systems (NIST CPS) Framework [1]. The CPS Framework makes it possible to link the low-level view of contracts to the high-level aspects of stakeholders' requirements and concerns about the supply chain as a whole. By integrating a multi-agent simulation with the structured approach of the NIST CPS Framework, we aim to capture the nuanced interplay of contractual obligations, agent behaviors, and their ramifications on the supply chain, providing insights into the potential points of failure and enabling strategies to improve resilience. The agents and the simulation environment are modeled using declarative techniques through Answer Set Programming (ASP) [3]. This approach allows for a clear specification of the logic governing agent behaviors and their contractual interactions. This paper presents not only the theoretical underpinnings of our simulation model but also a practical use case that illustrates its potential. Through these simulations, stakeholders can better understand the critical dependencies within their supply chains, evaluate the robustness of contractual arrangements, and explore strategies to enhance overall resilience.

2 A Distributed Multi-Agent Simulator

Two main components of the system are the environment simulator and the agent program. The code of the system is available on GitHub at `github.com/ltran1612/Research_CPS_SupplyChain`.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 125–127, doi:10.4204/EPTCS.416.10

Environment Simulator. The environment simulator consists of two main parts: the controller and the reasoning engine. The reasoning engine is a logic program that takes a state (of the environment) and a set of actions and determines whether the actions can be executed. In this case, the reasoning engine computes the state of the world. The environment simulator works with the global domain, denoted by D_{env}, which is the union of domains of all agents. The main portion of the code of the reasoning engine is for computing the effects of a set of actions on a state of the environment and is similar to the code for computing a plan in [5]. The controller is responsible for all the communication between the environment and the agents (e.g., receiving actions that agents execute, informing agents about the changes in the local state of agents). Besides disallowing the execution of incompatible actions, the simulator can also disrupt the execution of a supply chain by randomly disallowing the execution of some action. This forces the agents, whose actions are not executed, to replan with the new local state.

Agent Program. Each agent program consists of the following components: the action domain, the set of rules for reasoning about concerns of the agent, the set of rules for reasoning about the agent's actions as well as planning, and a controller. The controller is responsible for the communication of the agent with the environment. The controller's behaviour is described by the pseudocode given in Algorithm 1. The set of rules for reasoning about the agent's actions and planning is similar to the code for planning as described in [5].

Algorithm 1 Overall Behavior of the Agent Program

```
Require: agent ID, action domain, set of contracts
  registers ID with environment and wait for acknowledgement
  sends the action domain to the environment and wait for acknowledgement
  step = 0
  generate a plan p for the set of contracts
  while true do
      send p[step] (the step-th action of p) to environment
      wait for response (local_s – the local state) from the environment
      if local_s ≠ ⊥ then          ▷ all actions were executed successfully
          update the current state with local_s
          step = step + 1
      else                         ▷ some action cannot be executed
          generate a new plan p for the agent
          step = 0                 ▷ resetting
      end if
  end while
```

Communications. We used the publish/subscribe architecture (MQTT protocol specifically, see [6] for details) to facilitate the communications between agents and the environment. Each entity sends and receives messages by topics through a publish/subscribe broker component (middleman) [4]. Communication between an agent (with a unique *AgentID*) and the environment are done using two topics: "env/*AgentID*" and "for/*AgentID*". The former is for the agent to send and the latter is for receiving from the environment.

3 Case Study: Supply Chain Simulation

As a case study, we leveraged a dataset developed by the Observatory of Economic Complexity (OEC, `https://oec.world/en`), an organization that collects and analyzes international trade data. The dataset contains data about international imports and exports, from which we extracted part of an automotive supply chain. The scenario involves 8 agents with 7 contracts containing a total of 21 clauses, 8 supply chain requirements, and 5 concerns. An example of a contract clause from the scenario is: `C1: A responsible_for produced(vehicle_parts, 9K) when by_week 4`. Agent `A` is called speedy _auto_part. The private parts of the contract map `C1` to various requirements, addressing concerns of the CPS ontology. An example is: `C1: material-safe-for-production`. In this scenario, each agent can perform 4 types of actions: produce, deliver, pay, and receive. Each also keeps track of relevant fluents, like the total amount of payments made. The corresponding domains are encoded using a template. For example, action *produce* from `A` and related fluents are specified by rules including:

```
produce_item(speedy_auto_parts, vehicle_parts).
action(Agent, produce, (Item, Amount)) :- produce_item(Agent, Item), number(Amount).
causes(Agent, produce, Value, produced, increase, ()):-action(Agent, produce, Value).
```

A sample run of the simulator with the OEC agents progresses as follows. At first, the agents register with the environment and send their information (action domains and initial state) to the environment. At each step i, the agents send to the environment the actions they plan to execute. The environment computes the next state, sending back to the agents their local states, and waits for another set of actions. For example, at step 0, `A` has 0 `vehicle_part` and plans to take the action that produces 9,000 parts. The global state of the environment records at step 1 that `A` now has 9,000 `vehicle_part`, which is reflected in the local state of `A` as well. At the end of each iteration, the system determines which clauses are satisfied. For example, at step 0 no clauses are satisfied; at step 1, clause $C1$ is satisfied. In turn, the system uses this information to infer which requirements and concerns are satisfied.

4 Conclusions

In this paper, we presented a novel multi-agent simulation framework designed to address the complexities and challenges inherent in modern supply chains. We demonstrate that standard ASP programs such as planning or diagnosing modules can be integrated into a system for monitoring contract executions. By integrating the NIST CPS Framework with advanced contract formalization techniques, we have developed a robust system capable of simulating diverse supply chain scenarios and assessing the impact of various types of disruptions. Our evaluation on a realistic case study demonstrates that the framework not only enhances the understanding of supply chain dynamics but also provides actionable insights into improving resilience and reliability. It is understandable that a system utilizing ASP technology will inherit all of its problems (e.g., grounding, scalability). However, the prototype works fine with the use cases and appears acceptable to the owner of the use cases. Identifying the limit of the current system in terms of the limit of the system (e.g., the complexity of the domains or contracts) is an interesting issue and is our immediate future work.

Acknowledgement. Portions of this publication and research effort are made possible through the help and support of NIST via cooperative agreement 70NANB21H167.

References

[1] Edward, Christopher Greer, David A. Wollman & Martin J. Burns (2017): *Framework for cyber-physical systems: volume 1, overview*. doi:10.6028/NIST.SP.1500-201.

[2] Dylan Flynn, Chasity Nadeau, Jeannine Shantz, Marcello Balduccini, Tran Cao Son & Edward Griffor (2023): *Formalizing and Reasoning about Supply Chain Contracts between Agents*. In: 25th PADL, 13880, doi:10.1007/978-3-031-24841-2_10.

[3] Michael Gelfond & Vladimir Lifschitz (1991): *Classical Negation in Logic Programs and Disjunctive Databases*. New Generation Computing 9, pp. 365–385, doi:10.1007/BF03037169.

[4] B. Reselman (2021): *The pros and cons of the Pub-Sub architecture pattern*. Red Hat. Available at https://www.redhat.com/architect/pub-sub-pros-and-cons.

[5] Tran Cao Son, Enrico Pontelli, Marcello Balduccini & Torsten Schaub (2022): *Answer Set Planning: A Survey*. Theory and Practice of Logic Programming, pp. 1–73, doi:10.1017/S1471068422000072.

[6] M. Yuan (2021): *Getting to know MQTT*. IBM. Available at https://developer.ibm.com/articles/iot-mqtt-why-good-for-iot/.

Modular Stochastic Rewritable Petri Nets

Lorenzo Capra
Dipartimento di Informatica
Università degli Studi di Milano, Italy

Petri Nets (PN) are widely used for modeling concurrent and distributed systems, but face challenges in modeling adaptive systems. To address this, we have formalized 'rewritable' PT nets (RwPT) using `Maude`, a declarative language with sound rewriting logic semantics. Recently, we introduced a modular approach that utilizes algebraic operators to construct large RwPT models. This technique employs composite node labeling to outline symmetries in hierarchical organization, preserved through net rewrites. Once stochastic parameters are added to the formalism, we present an automated process to derive a *lumped* CTMC from the quotient graph generated by an RwPT.

1 Introduction

Traditional formalisms such as Petri Nets, Automata, and Process Algebra do not make it easy for designers to define dynamic system changes. Several extensions inspired by the π-calculus and the Nets-within-Nets paradigm have been proposed, but they often lack suitable analysis techniques. Rewritable PT nets (RwPT) [6] is versatile formalism for analyzing adaptive distributed systems. RwPT is specified using the declarative language `Maude`, which adopts Rewriting Logic to offer both operational and mathematical semantics, creating a scalable model for self-adapting PT nets. Unlike similar approaches ([2, 11]), the RwPT formalism provides data abstraction, is concise and efficient, and avoids the limitations posed by 'pushout' in Graph Transformation Systems. RwPT is an extension of GTS. Considering graph isomorphism (GI) when identifying equivalent states within the model dynamics is crucial to scaling up the model complexity. Recent work has shown that GI is quasi-polynomial [1]. Graph Canonization (GC) involves finding a representative such that for any two graphs G and G', $G \simeq G' \Leftrightarrow can(G) = can(G')$. We developed a general canonization technique [5] for use with RwPT, integrated into `Maude`. This technique works well for irregular models, but it is less effective for more realistic models.

In [7], we presented a technique for constructing comprehensive RwPT models using algebraic operators. The strategy is simple: leverage the modular characteristics of the models during analysis. By employing composite node labelling, we capture symmetries and sustain the hierarchical organization through net rewrites. A benchmark case study illustrates the effectiveness of our method. In this paper, we demonstrate a procedure to derive a lumped Continuous-Time Markov Chain (CTMC) from the quotient graph formed by an RwPT model, after introducing stochastic parameters into the framework.

The potential of `Maude` as functional logic programming framework was discussed in [10] and recently in https://logicprogramming.org/2023/02/extensions-of-logic-programming-in-maude/. Although we do not consider free variables in terms, our work can be seen as an example of *symbolic reachability* in which we use term rewriting through pattern matching (modulo normalization) instead of narrowing through unification. The use of narrowing to get a quotient state-transition system deserves further study.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 128–134, doi:10.4204/EPTCS.416.11

2 (Stochastic) PT nets, `Maude`, and demonstrative example

This section provides a concise overview of the (stochastic) PT formalism and emphasizes the key aspects of the `Maude` framework. For exhaustive information, we direct readers to the reference papers.

A *multiset* (or *bag*) b on a set D is a map $b : D \to \mathbb{N}$, where $b(d)$ is the *multiplicity* of d in b. We denote by $Bag[D]$ the set of multisets on D. Standard relational and arithmetic operations can be applied to multisets on a component-by-component basis. A stochastic PT (or SPN) *net* [12, 8] is a 6-tuple (P,T,I,O,H,λ), where: P, T are finite, non-empty, disjoint sets holding the net's nodes (places and transitions, respectively); $I,O,H : T \to Bag[P]$ represent the transitions' *input*, *output*, and *inhibitor* incidence matrices, respectively; $\lambda : T \to \mathbb{R}^+$ assigns each transition a negative exponential firing rate. A PT net *marking* (or state) is a multiset $m \in Bag[P]$.

The PT net dynamics is defined by the *firing rule*: $t \in T$ is *enabled* in m if $I(t) \leq m$ and $\forall p \in P : H(t)(p) = 0 \vee H(t)(p) > m(p)$. If t is enabled in m it may fire, leading to marking $m' = m + O(t) - I(t)$. We denote this: $m[t\rangle m'$. A PT-*system* is a pair (N,m_0), where N is a PT net and m_0 is a marking of N. The interleaving semantics of (N,m_0) is specified by the *reachability graph* (RG), an edge-labelled, directed graph (V,E) whose nodes are markings. It is defined inductively: $m_0 \in V$; if $m \in V$ and $m[t\rangle m'$ then $m' \in V$, $m \xrightarrow{t} m' \in E$. The timed semantics of a stochastic PT system is a CTMC isomorphic to the RG: For any two $m_i, m_j \in V$, the transition rate from m_i to m_j is $r_{i,j} := \sum_{t:m_i[t\rangle m_j} \lambda(t)$.

Maude `Maude` [9] is an expressive, purely declarative language characterized by a rewriting logic semantics [4]. Statements consist of *equations* and *rules*. Each side of a rule or equation represents terms of a specific *kind* that might include variables. Rules and equations have intuitive rewriting semantics, where instances of the left-hand side are substituted by corresponding instances of the right-hand side. The expressivity of `Maude` is realized through the use of matching modulo operator equational attributes, sub-typing, partiality, generic types, and reflection. A *functional* module comprises only *equations* and works as a functional program defining operations through equations, utilized as simplification rules. It details an *equational theory* within membership equational logic [3]: Formally, a tuple $(\Sigma, E \cup A)$, with Σ representing the signature, which includes the declaration of all sorts, subsorts, kinds[1], and operators; E being the set of equations and membership axioms; and A as the set of operator equational attributes (e.g., `assoc`). The model of $(\Sigma, E \cup A)$ is the *initial algebra* $T_{\Sigma/E \cup A}$, which mathematically corresponds to the quotient of the ground-term algebra T_Σ. Provided that E and A satisfy nonrestrictive conditions, the final (or *canonical*) values of ground terms form an algebra isomorphic to the initial algebra, that is, the mathematical and operational semantics coincide.

A `Maude` *system module* includes *rewrite rules* and, possibly, equations. Rules illustrate local transitions in a concurrent system. Specifically, a system module describes a generalized *rewrite theory* [4] $\mathscr{R} = (\Sigma, E \cup A, \phi, R)$, where $(\Sigma, E \cup A)$ constitutes a membership equation theory; ϕ identifies the frozen arguments for each operator in Σ; and R contains a set of rewrite rules. A rewrite theory models a concurrent system: $(\Sigma, E \cup A)$ establishes the algebraic structure of the states, while R and ϕ define the concurrent transitions of the system. The initial model of $\mathscr{R}$ assigns to each kind k a labeled transition system (TS) where the states are the elements of $T_{\Sigma/E \cup A,k}$, and transitions occur as $[t] \xrightarrow{[\alpha]} [t']$, with $[\alpha]$ representing *equivalent* rewrites. The property of *coherence* guarantees that a strategy that reduces terms to their canonical forms before applying the rules is sound and complete. A `Maude` system module is an executable specification of distributed systems. Given finite reachability, it enables the verification of invariant properties and the discovery of counterexamples.

[1] implicit equivalence classes defined by connected components of sorts (as per subsort partial order). Terms in a kind without a specific sort are *error* terms.

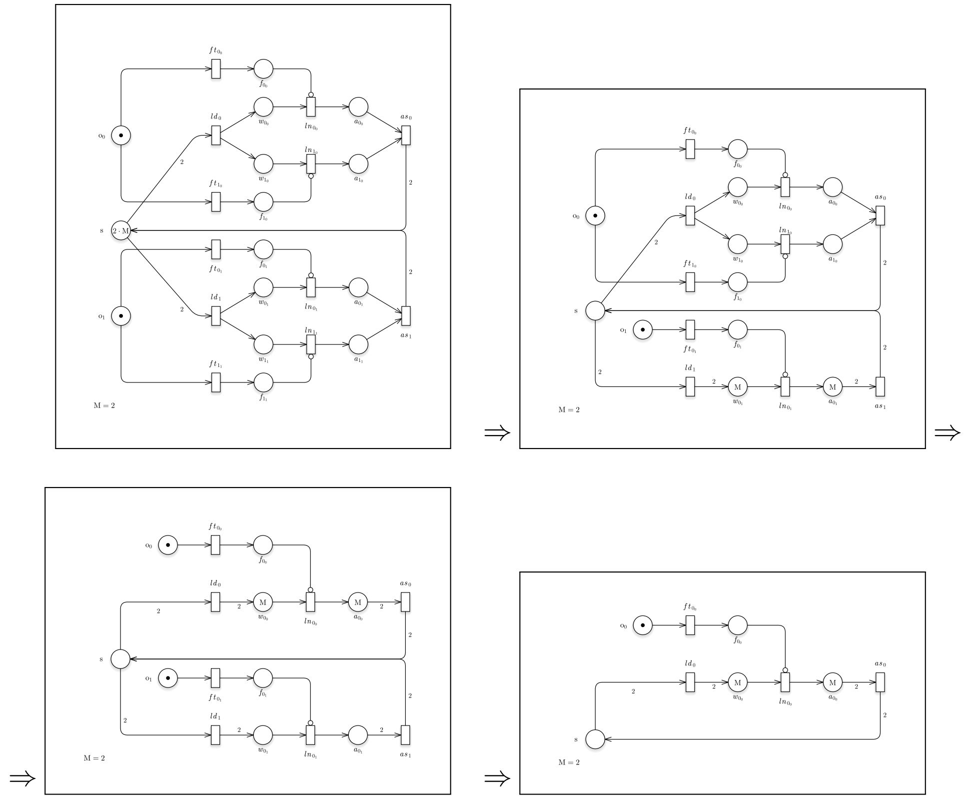


Figure 1: One of the possible paths of the Gracefully Degrading Production System.

Running example: fault-tolerant production line As an illustrative example, we refer to the model of a distributed production system that gracefully degrades presented in [7]. The system is composed of N production lines (PL), each branching into K fully interchangeable robots, that handle K raw items in parallel and assemble them into an artifact. In this study, $K = 2$. The items are loaded from a warehouse in an PL, K at a time. A robot in a PL might fail, upon which the PL restructures to continue functioning, but with reduced capacity. The reconfiguration process involves moving items from the faulted branch of a PL to the remaining branch(es) to maintain the production cycle. Traditional PNs are unable to model this operation. When a second fault occurs in a degraded PL, the system disconnects the PL. The leftover items are then relocated to the warehouse. Figure 1 shows the evolution of a system starting with two PLs. This scenario can be extended to N PLs, each operating K parallel robots, that handle $K \cdot M$ items (M being a third parameter of the model), denoted by the term `NPLsys(N, K, M)`.

3 Modular Rewritable Stochastic PN: Symmetries and Lumpability

Rewritable stochastic PT nets (RwSPT) build upon the concept of *modular* rewritable PT nets [7] by linking negative exponential rates to the firing of PT transitions and the process of net rewrites.

The definition of RwSPT includes a hierarchy of **Maude** modules (e.g., `BAG`, `PT-NET`, `PT-SYSTEM`) described in [7]. The `Maude` sources can be found in https://github.com/lgcapra/rewpt/tree/main/modSPT. RwSPT uses structured annotations to underline the symmetry of the model. It features a concise place-based encoding to aid in state canonization and is based on the functional module `BAG{X}`, which introduces multisets as a complex data type. The commutative and associative `_+_` operator provides an intuitive way to describe a multiset as a weighted sum, for instance, `3 . a + 1 . b`. The sort `Pbag` contains multisets of places. Each place label (a term of sort `Plab`) is a nonempty list of pairs built of `String` and a `Nat`. Places are uniquely identified by their labels. These pairs represent a symmetric component within a nested hierarchy. Compositional operators annotate places incrementally from right to left: The label suffix represents the root of a hierarchy. For example, the 'assembly' place of line 1 in Production Line 2 would be encoded as: `p(< "a"; 0 > < "L"; 1 >)`. We implicitly describe net transitions (terms `Tran`) through their incidence matrix (a 3-tuple of terms `Pbag`) and associated tags. A tag includes a descriptive `String` and a `Float` interpreted as a firing rate. The syntax is: `[I,O,H] -> << S, R >>`.

Using the associative composition operator `_;_` and the subsort relation `Tran < Net`, we can easily construct PT nets in a modular way. For example, we can depict the subnet containing the load transition (*ld*) and a robot (ln_0) as the `Net` term in the listing below.

```
[2 . p(< "s" ; 0 >), 1 . p(< "w" ; 0 >) + 1 . p(< "w" ; 1 >), nilP] |-> << "ld", 0.5 >> ;
[1 . p(< "w" ; 0 >), 1 . p(< "a" ; 0 >), 1 . p(< "f" ; 0 > ] |-> << "ln", 0.1 >>
```

A `System` term is the empty juxtaposition (`__`) of a `Net` and a `Pbag` representing the marking. The conditional rewrite rule `firing` specifies the PT firing rule (notice the use of a matching equation :=).

```
vars N N' : Net . vars T : Tran . var M : Pbag .
crl [firing] : N M => N firing(T, M) if T ; N' := N /\ enabled(T, N M) .
```

An RwSPT is defined by a system module that contains two constant operators, used as aliases: `op net : -> Net` and `op m0 : -> Pbag .` Two equations define their bindings. This module includes a set R of `System` rewrite rules incorporating `firing`. We adopt interleaving semantics: Rewrites have an exponential rate (specified in the rule label but for `firing`), so that for the state transition system it holds ($\subseteq$ is the subgraph relation): $TS(\texttt{net m0}, \{\texttt{firing}\}) \subseteq TS(\texttt{net m0}, R)$.

Modularity, symmetries, and lumpability We have provided net-algebra and net-rewriting operators [7] with a twofold intention: to ease the modeler's task and to enable the construction and modification of large-scale models with nested components by implicitly highlighting their symmetry. A compact *quotient* TS is built using simple manipulation of node labels. This approach outperforms that integrated into `Maude` [5] and based on traditional graph canonization.

In this context, the identification of behavioral equivalences is reduced to a graph *morphism*. PT system morphism must maintain the edges and the marking: In our encoding, a *morphism* between PT systems `(N m)` and `(N' m')` is a bijection ϕ : `places(N)` $\rightarrow$ `places(N')` such that, considering the homomorphic extension of ϕ on multisets, $\phi(\texttt{N}) = \texttt{N'}$ and $\phi(\texttt{m}) = \texttt{m'}$. Moreover, ϕ must retain the textual annotations of the place labels and the transition tags. If `N' = N` we speak of *automorphism*, in which case ϕ is a permutation in the set of places. We refer to a *normal* form that principally involves identifying sets of automorphic (permutable) places: Two markings `m`, `m'` of a net `N` are said automorphic if there is an automorphism ϕ in `N` that maps `m` into `m'`. We denote this `m` $\cong$ `m'`. The equivalence relation $\cong$ is a congruence, that is, it preserves the transition firings and *rates*.

Definition 3.1 (Symmetric Labeling)**.** *A* `Net` *term is symmetrically labeled if any two maximal sets of places whose labels have the same suffix (possibly empty), which is preceded by pairs with the same tag, are permutable. A* `System` *term is symmetrically labeled if its* `Net` *subterm is.*

In other words, if a `Net` term `N` meets definition 3.1, then for any two maximal subsets of places matching:

$P := \{\texttt{p(L' < w ; i > L)}\}$, $\quad P' := \{\texttt{p(L'' < w ; j > L)}\}$,

where: `L, L', L'' : Plab, w: String, i, j : Nat` there is an automorphism ϕ such that $\phi(P) = P'$, $\phi(P') = P$, which is extended as an identity to the rest[2].

If a `System` term adheres to the previous definition, it can be transformed into a 'normal' form by merely swapping indices on the place labels (e.g., i $\leftrightarrow$ j), while still complying with definition 3.1. This normal form is the most minimal according to a lexicographic order within the automorphism class ($\cong$) implicitly defined by 3.1. In contrast to general graph canonization, there is no need for any pruning strategy or backtracking: A monotone procedure is used where the sequence of index swaps does not matter (see [7]). Efficiency is achieved as the normalized form of the subterm `Net` is derived through basic "name abstraction", where at each hierarchical level the indices of structured place labels continuously span from 0 to $k \in \mathbb{N}$.

The strategy involves providing a concise set of operators that preserve nets' symmetric labelling. This set includes *compositional* operators and operators for *manipulating* nets. Rewrite rules require these operators to manipulate `System` terms defined in a modular manner. Furthermore, rules must adhere to parametricity conditions that limit the use of ground terms [7]. Under these assumptions, we get a *quotient* TS from a `System` term that retains reachability and meets strong bisimulation.

Let t, t', u, u' be (final) terms of sort `System`, r a `System` rule $r: \ s \Rightarrow s'$. The notation $t \overset{r(\sigma)}{\Rightarrow} t'$ means there is a ground substitution σ of r's variables such that $\sigma(s) = t$ and $\sigma(s') = t'$.

Property 3.1. *Let t meet Definition 3.1. If $t \overset{r(\sigma)}{\Rightarrow} t'$, then $\forall u, \phi, t \cong_{\phi} u$: $u \overset{r(\phi(\sigma))}{\Rightarrow} u'$, $t' \cong u'$*

The TS quotient generated by a normal form $\hat{t}$ is obtained by applying the operator `normalize` to the terms on the right side of the rewrite rules. When a `System` undergoes a rewrite due to the `firing` rule, the process only involves the marking subterm. According to property 3.1 (firing preservation), because the morphism ϕ preserves the transition rates and the rules are parameterized, it is feasible to map the TS quotient of $\hat{t}$ onto an isomorphic "lumped" CTMC: In a Markov process's state space, an equivalence relation is considered "strong lumpability" if the cumulative transition rates between any two states within a class to any other class remain consistent. Despite the possibility of establishing a more stringent condition, namely "exact lumpabability," we focus on the aggregated probability.

4 Getting the Lumped CTMC generator from RwSPT

A rewritable PT `system` generates a transition system (TS) isomorphic to a lumped CTMC. However, the TS produced via the `show search graph` command of `Maude` embodies a *parametric* CTMC: in line with the rewriting logic semantics, state transitions denote *classes* of equivalent rewrites, meaning, PT firings that result in identical normalized markings or net rewrites that lead to isomorphic PT systems.

To obtain the CTMC generator, it is necessary to quantify instances that align with a specific state transition. Regrettably, the `Maude` system lacks a mechanism to determine the matches of a rewrite rule in the TS construction process.

Our solution consists of first (automatically) generating a Transition System with states having a composite structure, which provides a detailed view of the equivalent rewrites that result in state transitions. Subsequently, using elementary parse to compute the cumulative rates of the lumped CTMC. Despite a redundant state representation, this method incurs an acceptable time overhead because it only involves *normalized* states.

When considering the term `NPLsys(2,2,2)`, which aligns with the PT net at the top left of Figure 1, the resulting quotient TS comprises 295 states compared to the 779 states in the standard TS. State transitions often correspond to multiple matches: For instance, the initial state (the term above) includes

[2]According to the definition of PT morphism, the prefixes `L'` and `L''` are consistent in the textual component.

two 'load' instances and four 'fault' instances that lead to markings with identical normal forms. Consequently, the combined rates are $2 \cdot 0.5$ and $4 \cdot 0.001$.

Experimental Evidence We showcase experimental validation of the method and a demonstration of standard performance indicators. Table 1 displays the results of the `search` command to locate the final states. We used Linux WSL on an 11th Gen. Intel Core i5 with 40GB RAM. The state spaces match those of the lumped CTMC. The analysis of large models is feasible solely by exploiting the model's symmetry. Notice that the number of absorbing states in the TS quotient does not vary with N.

Figure 2 shows the system *reliability* (the complement of Time to System Failure distribution). As expected, is decreases with time; additionally, the scenario that involves more replicas demonstrates enhanced reliability. The inflexion point at around time 800 represents the system's reconfiguration time. The increased execution time of the job (not reported) is a result of a system failure. The overall trend is also noticeable when we look at larger values of `N`. As `N` increases, both reliability and throughput curves show significant improvements. However, we observe an asymptotic trend when `N` is greater than 6. Our interpretation is that beyond a certain point, the benefit of using a higher number of replicas is outweighed by the higher fault rate and the increased configuration overhead.

Table 1: Ordinary vs Quotient TS of `NPLsys(N,2,2)` † `search` timed out after 10 h

	Ordinary		Quotient	
`N`	states(final)	time (sec)	states(final)	time (sec)
1	60(2)	0	42(2)	0
2	779(4)	0.1	295(2)	0.1
3	6101(6)	4.8	1059(2)	0.9
4	37934(8)	69	2764(2)	3.6
5	204362(10)	818	5970(2)	10
6	1000187(12)	13930	11367(2)	27
7	-	†	19775(2)	65
8	-	†	32144(2)	186
9	-	†	49554(2)	569
10	-	†	73215(2)	2450

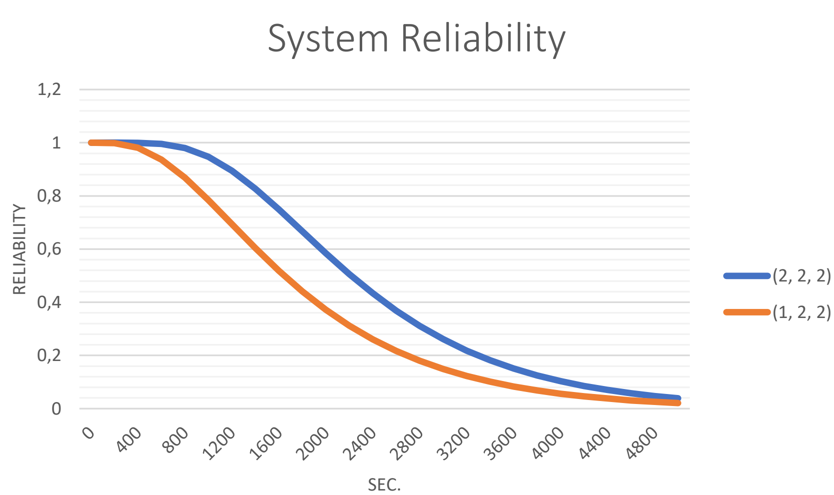


Figure 2: System Reliability.

5 Conclusion and Future Work

We have developed a lumped Markov process for modular and rewritable Petri nets (RwPT), a flexible model of adaptive distributed systems. RwPTs, which we construct and manipulate using a small set of algebraic operators, exhibit structural symmetries that result in an efficient quotient state-transition graph. We have outlined a semi-automatic procedure for deriving the CTMC infinitesimal generator from the RwPT quotient graph. Future efforts will focus, on the one hand, on exploring orthogonal structured solutions and, on the other, on fully implementing the process and integrating it into graphical editors. We aim to broaden the approach to derive a lumped Markov process from any `Maude` specification.

References

[1] László Babai (2016): *Graph Isomorphism in Quasipolynomial Time [Extended Abstract]*. In: *Proceedings of the Forty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '16, Association for Computing Machinery, New York, NY, USA, p. 684–697, doi:10.1145/2897518.2897542.

[2] Paulo E. S. Barbosa, João Paulo Barros, Franklin Ramalho, Luís Gomes, Jorge Figueiredo, Filipe Moutinho, Anikó Costa & André Aranha (2011): *SysVeritas: A Framework for Verifying IOPT Nets and Execution Semantics within Embedded Systems Design*. In Luis M. Camarinha-Matos, editor: *Technological Innovation for Sustainability - Second IFIP WG 5.5/SOCOLNET Doctoral Conference on Computing, Electrical and Industrial Systems, DoCEIS 2011, Costa de Caparica, Portugal, 2011. Proceedings*, *IFIP Advances in Information and Communication Technology* 349, Springer, pp. 256–265, doi:10.1007/978-3-642-19170-1_28.

[3] Adel Bouhoula, Jean-Pierre Jouannaud & José Meseguer (2000): *Specification and proof in membership equational logic*. *Theoretical Computer Science* 236(1), pp. 35–132, doi:10.1016/S0304-3975(99)00206-6.

[4] Roberto Bruni & José Meseguer (2003): *Generalized Rewrite Theories*. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow & Gerhard J. Woeginger, editors: *Automata, Languages and Programming*, Springer-Verlag, Berlin, Heidelberg, pp. 252–266, doi:10.1007/3-540-45061-0_22.

[5] Lorenzo Capra (2022): *Canonization of Reconfigurable PT Nets in Maude*. In Anthony W. Lin, Georg Zetzsche & Igor Potapov, editors: *Reachability Problems*, Springer International Publishing, Cham, pp. 160–177, doi:10.1007/978-3-031-19135-0_11.

[6] Lorenzo Capra (2022): *Rewriting Logic and Petri Nets: A Natural Model for Reconfigurable Distributed Systems*. In Raju Bapi, Sandeep Kulkarni, Swarup Mohalik & Sathya Peri, editors: *Distributed Computing and Intelligent Technology*, Springer International Pub., pp. 140–156, doi:10.1007/978-3-030-94876-4_9.

[7] Lorenzo Capra & Michael Köhler-Bußmeier (2024): *Modular rewritable Petri nets: An efficient model for dynamic distributed systems*. *Theoretical Computer Science* 990, p. 114397, doi:10.1016/j.tcs.2024.114397.

[8] Giovanni Chiola, Marco Ajmone Marsan, Gianfranco Balbo & Gianni Conte (1993): *Generalized Stochastic Petri Nets: A Definition at the Net Level and Its Implications*. *IEEE Trans. Software Eng.* 19, pp. 89–107, doi:10.1109/32.214828.

[9] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso M. Oliet, Jos'e Meseguer & Carolyn Talcott (2007): *All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic*. LNCS, Springer, doi:10.1007/978-3-540-71999-1.

[10] Santiago Escobar (2014): *Functional Logic Programming in Maude*, pp. 315–336. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-54624-2_16.

[11] Julia Padberg & Alexander Schulz (2016): *Model Checking Reconfigurable Petri Nets with Maude*. In Rachid Echahed & Mark Minas, editors: *Graph Transformation*, Springer International Publishing, Cham, pp. 54–70, doi:10.1007/978-3-319-40530-8_4.

[12] W. Reisig (1985): *Petri Nets: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, doi:10.1007/978-3-642-69968-9.

Semantic Argumentation using Rewriting Systems

Esteban Guerrero
Department of Computing Science, Umeå University, Sweden
esteban.guerrero@umu.se

Juan Carlos Nieves
Department of Computing Science, Umeå University, Sweden
juan.carlos.nieves@umu.se

In this article, we introduce a general framework for *structured argumentation* providing consistent and well-defined *justification* for conclusions that *can* and *cannot* be inferred and there is certainty about them, which we call semantic and NAF-arguments, respectively. We propose the so-called *semantic argumentation* guaranteeing well-known principles for quality in structured argumentation, with the ability to generate semantic and NAF-arguments, those where the *conclusion* atoms are semantically interpreted as *true*, and those where the conclusion is assumed to be *false*. This framework is defined on the set of all logic programs in terms of *rewriting systems* based on a *confluent set* of *transformation rules*, the so-called *Confluent Logic Programming Systems*, making this approach a general framework. We implement our framework named *semantic argumentation solver* available open source.

1 Motivation

We propose a general argument construction based on the partial interpretation of programs using different *families* of *logic programming semantics* induced by *rewriting systems functions* [6]. *Rewriting rules* are used to replace parts of a logic program based on the concept of a *normal form*, which is the least expression of a program that cannot be rewritten any further [9]. For example, having a program with the only rule: $innocent(x) \leftarrow \ not\ guilty(x)$, current structured argumentation approaches [10] generate the only consistent argument: $\langle \underbrace{\{innocent(X) \leftarrow \ not\ guilty(x)\}}_{Support}, \underbrace{innocent(x)}_{Conclusion} \rangle$, expressing that person x is innocent if x can not be proved guilty. However, in domain applications that need the generation of argument-based *reason explanations*, providing structured and well-defined reasons why x is not guilty ($not\ guilty(x)$) are needed. We emphasize the role of investigating such computational mechanisms that can also build arguments justifying conclusions based on the atoms that are inferred as *false*, *e.g.*, to state that *there is certainty in affirming that the guiltiness of x is false (there is no evidence), therefore the x must be innocent*, *i.e.*, $\langle \underbrace{\{innocent(X) \leftarrow \ not\ guilty(x)\}}_{Support}, \underbrace{not\ guilty(x)}_{Conclusion} \rangle$. These types of arguments have been less explored in the formal argumentation theory, except for *assumption-based argumentation* (ABA) [8].

2 Syntax and semantics

We use propositional logic with the following connectives $\wedge, \leftarrow$, *not*, and $\top$ where $\wedge$, and $\leftarrow$ are 2-place connectives, *not* and $\top$. The negation symbol *not* is regarded as the so-called *negation as failure* (NAF).

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 135–138, doi:10.4204/EPTCS.416.12

We follow standard logic programming syntax, *e.g.*, [5], for lack of space we do not include some basic and well-established syntax.

An *interpretation* of the signature $\mathcal{L}_P$ is a function from $\mathcal{L}_P$ to {false,true}. A *partial interpretation* of $\mathcal{L}_P$, are the sets $\langle I_1, I_2\rangle$ where $I_1 \cup I_2 \subseteq \mathcal{L}_P$. We use $\mathtt{SEM}(P) = \langle P^{true}, P^{false}\rangle$, where $P^{true} := \{p \mid p \leftarrow \top \in P\}$ and $P^{false} := \{p \mid p \in \mathcal{L}_P \backslash \mathtt{HEAD}(P)\}$. $\mathtt{SEM}(P)$ is also called model of P [6]. We use three value semantics that are characterized by ***rewriting systems*** following a set of **Basic Transformation Rules** for **Rewriting Systems** (see details in [6]), those rules are named: **RED+**, **RED-**, **Success**, **Failure**, **Loop**, **SUB**, and **TAUT**. Then, two rewriting systems ($\mathcal{CS}$) can be defined based on the previous basic transformations: $CS_0 = \{RED^+, RED^-, Success, Failure, Loop\}$, induces the WFS [3]. $CS_1 = CS_0 \cup \{SUB, TAUT, LC\}$, induces WFS+ [6]. The *normal form* of a normal logic program P with respect to a rewriting system $\mathcal{CS}$ is denoted by $norm_{\mathcal{CS}}(P)$. Every rewriting system $\mathcal{CS}$ induces a 3-valued logic semantics $\mathtt{SEM}_{\mathcal{CS}}$ as $\mathtt{SEM}_{\mathcal{CS}}(P) := \mathtt{SEM}(norm_{\mathcal{CS}}(P))$. To simplify the presentation, we use the entailment $\models_{\mathtt{SEM}_{\mathcal{CS}}}$ applied to a logic program P is defined by $\mathtt{SEM}_{\mathcal{CS}}(P) = \langle T, F\rangle$ in which $P \models_{\mathtt{SEM}^T_{\mathcal{CS}}} a$ if and only if $a \in T$, similarly, if $P \models_{\mathtt{SEM}^F_{\mathcal{CS}}} a$ if and only if $a \in F$. We use the entailment $\models_{\mathtt{SEM}_{\mathcal{CS}_0}}$ and $\models_{\mathtt{SEM}_{\mathcal{CS}_1}}$ for confluent rewriting system CS_0 and CS_1 respectively; and the form $\models_{\mathtt{SEM}_{\mathcal{CS}}}$ to indicate that any rewriting system can be used.

3 Semantic and NAF-arguments

Let us introduce a formal definition of semantic arguments.

Definition 1 (Semantic argument) *Given a normal logic program P and $S \subseteq P$. $Arg_P = \langle S,\ g\rangle$ is a* ***semantic argument*** *under $\mathtt{SEM}_{\mathcal{CS}}$ w.r.t. P, if the following conditions hold true:*

1. $S \models_{\mathtt{SEM}^T_{\mathcal{CS}}} g$
2. *S is minimal w.r.t. the set inclusion satisfying 1.*

We simplify the notation of these semantic arguments as $\mathcal{A}rg^+$. Condition 1 states that the interpretation of conclusion g is *true w.r.t.* T in $\mathtt{SEM}_{\mathcal{CS}}(S)$. Condition 2 in Definition 1 guarantees the support minimality.

Now, let us define NAF-arguments as follows:

Definition 2 (NAF-arguments) *Given a normal logic program P and $S \subseteq P$. $Arg_P = \langle S,\ not\ g\rangle$ is a* ***NAF-argument*** *under the $\mathtt{SEM}_{\mathcal{CS}}$ w.r.t. P, if the following conditions hold true:*

1. $S \models_{\mathtt{SEM}^F_{\mathcal{CS}}} g$,
2. *S is minimal w.r.t. the set inclusion satisfying 1.*

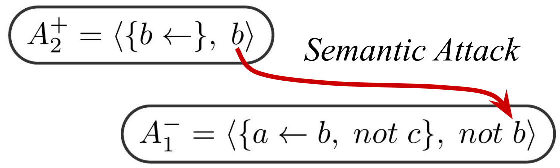
Condition 1 in Definition 2 is the interpretation of the conclusion *w.r.t.* $\models_{\mathtt{SEM}^F_{\mathcal{CS}}}$, with the set of all the NAF-arguments noted as $\mathcal{A}rg^-$. The addition of *not* in the conclusion of a NAF-argument stresses that such an atom is interpreted as false by $\mathtt{SEM}_{\mathcal{CS}}$.

Example 1 *Let us consider a program P_3 for building semantic and NAF-arguments considering $\mathcal{CS}_0$ and $\mathcal{CS}_1$.*

We build semantic and NAF-arguments as follows: 1) get related clauses of atoms (S_i); 2) for every related clause compute $\mathtt{SEM}_{\mathcal{CS}_0}(S_i)$ and $\mathtt{SEM}_{\mathcal{CS}_1}(S_i)$; 3) the support (every S_i) is joined to the conclusion[1]. Then, the following sets of arguments are built considering $\mathcal{CS}_0$ and $\mathcal{CS}_1$:

[1]We implemented this procedure and the sources are open, then can be found in `https://people.cs.umu.se/~esteban/argumentation/`

$$P_3 := \left\{ \begin{array}{l} a \leftarrow b, not\ c \\ b \leftarrow \\ d \leftarrow not\ d \\ e \leftarrow e \end{array} \right\}$$



Case $\mathscr{CS}_0$: $\mathscr{A}rg_{P_3} = \{A_1^+, A_2^+, A_3^+, A_1^-, A_2^-, A_3^-, A_4^-, A_6^-\}$.
Case $\mathscr{CS}_1$: $\mathscr{A}rg_{P_3} = \{A_1^+, A_2^+, A_3^+, A_5^+, A_1^-, A_2^-, A_3^-, A_4^-, A_6^-\}$.

An effect of interpreting argument supports under $\texttt{SEM}_{\mathscr{CS}}$ is that some atoms (or sets of them) are evaluated in *opposition* to other arguments (*e.g.*, $A_1^+ = \langle S_2, a \rangle$ and $A_1^- = \langle S_1,\ not\ a \rangle$ in Example 1), suggesting a *semantic attack* relationship.

Definition 3 (Semantic attack) *Let $A = \langle S_A, a \rangle \in \mathscr{A}rg^+$, $B = \langle S_B, not\ b \rangle \in \mathscr{A}rg^-$ be two semantic arguments where $\texttt{SEM}_{\mathscr{CS}}(S_A) = \langle T_A, F_A \rangle$ and $\texttt{SEM}_{\mathscr{CS}}(S_B) = \langle T_B, F_B \rangle$. We say that A attacks B if $x \in T_A$ and $x \in F_B$, denoted $\texttt{attacks}(x, y)$.*

Lemma 1 *Semantic and NAF-arguments built from any normal logic program are always consistent.*

Definition 4 (Semantic Argumentation Framework (SAF)) *Let P be a normal program. Then, a **semantic argumentation framework** is the tuple: $SAF_P = \langle \mathscr{A}rg_P, \mathscr{A}tt \rangle$*

We can straightforward extend the definitions of *argumentation semantics* in [7] as follows:

Definition 5 *Let $SAF_P = \langle \mathscr{A}rg_P, \mathscr{A}tt \rangle$ be a semantic argumentation framework. An admissible set of arguments $S \subseteq AR$ is:*

- stable *if and only if S attacks each argument which does not belong to S.*
- preferred *if and only if S is a maximal (*w.r.t. *inclusion) admissible set of AF.*
- complete *if and only if each argument, which is acceptable with respect to S, belongs to S.*
- grounded *if and only if S is the minimal (*w.r.t. *inclusion)* complete *extension of AF*[2].

Example 2 *Let us consider $P_5 = \{a \leftarrow not\ b;\ b \leftarrow not\ a;\ c \leftarrow not\ c, not\ a;\ d \leftarrow not\ d, not\ b;\}$. $\texttt{SEM}_{\mathscr{CS}}$ will remove rules involving atoms c and d. Then, applying Definition 4, we have the framework: $SAF_{P_5} = \langle \{A_6^- = \langle \{a \leftarrow not\ b\},\ not\ a \rangle, A_6^+ = \langle \{b \leftarrow not\ a\},\ b \rangle,\ A_5^- = \langle \{b \leftarrow not\ a\},\ not\ b \rangle,\ A_5^+ = \langle \{a \leftarrow not\ b\},\ a \rangle\},\ \texttt{attacks}(A_5^+, A_6^+),\ \texttt{attacks}(A_5^+, A_6^-),\ \texttt{attacks}(A_6^+, A_5^+),\ \texttt{attacks}(A_6^+, A_5^-) \rangle$. When we apply Definition 5 to SAF_{P_5} we obtained the following extensions:*

- *Stable = preferred:* $\{\{A_5^+, A_5^-\},\ \{A_6^+, A_6^-\}\}$
- *Complete:* $\{\{A_5^+, A_5^-\},\ \{A_6^+, A_6^-\},\ \{\}\}$
- *Grounded:* $\{\}$

4 Conclusions

The main contributions are: 1) *Semantic Argumentation Frameworks* (SAF) can be used for justifying *true* and *false* interpreted conclusions. 2) SAF is based on *families* of rewriting confluent systems. 3) Satisfies all the well-known argumentation postulates [1, 4]. Future work will involve the exploration of our framework under other Confluent Logic Programming Systems, the satisfaction of other argumentation principles, and the investigation of commonalities between ABA and semantic argumentation.

[2] In [2] it is shown that grounded can be characterized in terms of complete extensions.

References

[1] Leila Amgoud (2014): *Postulates for logic-based argumentation systems*. International Journal of Approximate Reasoning 55(9), pp. 2028–2048, doi:10.1016/j.ijar.2013.10.004.

[2] Pietro Baroni, Martin Caminada & Massimiliano Giacomin (2011): *An introduction to argumentation semantics*. The knowledge engineering review 26(4), pp. 365–410, doi:10.1017/S0269888911000166.

[3] Stefan Brass, Ulrich Zukowski & Burkhard Freitag (1997): *Transformation-based bottom-up computation of the well-founded model*. In: Non-Monotonic Extensions of Logic Programming, Springer, pp. 171–201, doi:10.1007/BFb0023807.

[4] Martin Caminada & Leila Amgoud (2007): *On the evaluation of argumentation formalisms*. Artificial Intelligence 171(5-6), pp. 286–310, doi:10.1016/j.artint.2007.02.003.

[5] Jürgen Dix (1995): *A Classification Theory of Semantics of Normal Logic Programs: I. Strong Properties.* Fundam. Inform. 22(3), pp. 227–255, doi:10.3233/FI-1995-2234.

[6] Jürgen Dix, Mauricio Osorio & Claudia Zepeda (2001): *A general theory of confluent rewriting systems for logic programming and its applications*. Annals of Pure and Applied Logic 108(1-3), pp. 153–188, doi:10.1016/S0168-0072(00)00044-0.

[7] Phan Minh Dung (1995): *On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and n-Person Games*. Artificial Intelligence 77(2), pp. 321–358, doi:10.1016/0004-3702(94)00041-X.

[8] Phan Minh Dung, Robert A. Kowalski & Francesca Toni (2009): *Assumption-Based Argumentation*. In: Argumentation in Artificial Intelligence, Springer, pp. 199–218, doi:10.1007/978-0-387-98197-0_10.

[9] Juan Carlos Nieves & Mauricio Osorio (2016): *Ideal extensions as logical programming models*. Journal of Logic and Computation 26(5), pp. 1361–1393, doi:10.1093/logcom/exu014.

[10] Henry Prakken (2010): *An abstract framework for argumentation with structured arguments*. Argument and Computation 1(2), pp. 93–124, doi:10.1080/19462160903564592.

Data2Concept2Text: An Explainable Multilingual Framework for Data Analysis Narration*

Flavio Bertini Alessandro Dal Palù Federica Zaglio
Department of Mathematical, Physical and Computer Sciences, University of Parma,
Parma, Italy
{flavio.bertini|alessandro.dalpalu|federica.zaglio}@unipr.it

Francesco Fabiano
Department of Computer Science, New Mexico State University
Las Cruces, New Mexico, USA
ffabiano@nmsu.edu

Andrea Formisano
Department of Mathematics, Computer Science and Physics, University of Udine
Udine, Italy
andrea.formisano@uniud.it

This paper presents a complete explainable system that interprets a set of data, abstracts the underlying features and describes them in a natural language of choice. The system relies on two crucial stages: (i) identifying emerging properties from data and transforming them into abstract concepts, and (ii) converting these concepts into natural language. Despite the impressive natural language generation capabilities demonstrated by Large Language Models, their statistical nature and the intricacy of their internal mechanism still force us to employ these techniques as black boxes, forgoing trustworthiness.

Developing an explainable pipeline for data interpretation would allow facilitating its use in safety-critical environments like processing medical information and allowing non-experts and visually impaired people to access narrated information. To this end, we believe that the fields of knowledge representation and automated reasoning research could present a valid alternative. Expanding on prior research that tackled the first stage (i), we focus on the second stage, named *Concept2Text*. Being explainable, data translation is easily modeled through logic-based rules, once again emphasizing the role of declarative programming in achieving AI explainability.

This paper explores a Prolog/CLP-based rewriting system to interpret concepts—articulated in terms of classes and relations, plus common knowledge—derived from a generic ontology, generating natural language text. Its main features include hierarchical tree rewritings, modular multilingual generation, support for equivalent variants across semantic, grammar, and lexical levels, and a transparent rule-based system. We outline the architecture and demonstrate its flexibility through some examples capable of generating numerous diverse and equivalent rewritings based on the input concept.

1 Introduction

The emergence of explainable Artificial Intelligence (xAI) signifies the integration of crucial aspects within AI systems, such as transparency, ethical conduct, accountability, privacy, and fairness [2]. In

*Research partially supported by Interdepartmental Project on AI (Strategic Plan UniUD–22-25) and by INdAM–GNCS project CUP E53C22001930001. F.Bertini, A.Dal Palù, and A.Formisano are members of GNCS-INdAM, Gruppo Nazionale per il Calcolo Scientifico.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 139–152, doi:10.4204/EPTCS.416.13

several domains, the acceptance of AI systems depends on their ability to offer comprehensive insights into their internal workings and transparency in decision-making processes. Notably, the recent European Union AI Act aims to establish a unified legal framework to foster AI development while safeguarding public interests, such as health, safety, fundamental rights, democracy, and the environment [7]. This legislation mandates AI systems to be sufficiently transparent, explainable, and well-documented, necessitating them to provide supporting evidence for their outputs. These considerations are especially important when AI systems are utilized in high-risk scenarios, such as automatically describing an electrocardiogram in a medical report. While achieving these objectives remains challenging for systems reliant on deep neural networks, it presents an opportunity for the Logic Programming community due to the inherent explainability of its products.

A simple count on papers on xAI classified by Scopus per year was analyzed by the system presented in this paper. The following is one automatically generated output:

```
From the year 2014 up to 2023 publications in explainable AI have
    exponentially grown in an important way (from 0 up to 1905) [excellent
    accuracy]; in detail, during the interval of time between the years 2014
    and 2017 publications have been significantly steady (from 0 to 7)
    [excellent accuracy].
```

Our research focuses on transforming raw data, such as data series, into natural language descriptions within an explainable framework. This involves interpreting and abstracting features (*Data2Concept*) and subsequently translating concepts into natural language (*Concept2Text*). The first step, as shown in [4, 5], requires identifying user-defined patterns in raw data and representing them as high-level descriptions or *concepts*. For instance, a time series showing a consistent increase in values over time can serve as a logical fact for subsequent processing, enriched with additional contextual information. Our previous work primarily addressed the second step, i.e., Concept2Text, demonstrating a trivial natural language expression generator. Extensions of this work have led to domain-specific applications, such as an explainable decision-making support system for analytics in academia [3].

In this work, instead, we focus on the design of a general Concept2Text pipeline whose key features include:

- **Explainability.** Our system is grounded in Logic Programming and focuses on rewriting rules and Constraint Satisfaction Problems providing transparency at every level.
- **Modularity.** The system allows for seamless expansion to accommodate various domain-specific concepts and languages thanks to its declarative nature.
- **Tree Rewriting.** We represent concepts as trees that are manipulated progressing from the conceptual to the the syntax level generating natural language.
- **Variants.** To facilitate the generation of diverse semantic-equivalent sentences, each rewriting permits the creation of multiple versions that can be selected. Each stage of rewriting determines various levels of equivalence, ranging from conceptual and structural to grammatical and lexical.
- **Multi-Language Support.** The modular architecture of the system enables the creation of multiple language rule sets without affecting the overall structure. In this paper, this is demonstrated with examples in English and Italian.

The paper is structured as follows. Section 2 provides a concise overview of the background. Section 3 presents the design of the system, while Section 4 demonstrates some practical results. Finally, Section 5 offers concluding remarks.

2 Background

Ontologies In information science, ontologies act as pivotal organizational frameworks, structuring knowledge within defined domains by delineating facts, properties, and their interconnections via representational elements such as classes, attributes, and relations among them [20]. These conceptual constructs not only explain the intricate relationships between pertinent concepts in a domain but also capture knowledge across multiple domains from the arts and sciences to cutting-edge technologies and medical sciences.

Practically, ontologies use specific languages to articulate concepts and relationships, removing the complexities of implementation. One notable exemplar in this realm is the Web Ontology Language (OWL), which allows applications to process information and concepts in autonomy [13]. Ontologies created a profound transformation in computational reasoning, equipping machines with the capability to decipher word meanings and assemble them into intricate sentences, similar to natural language processing [19].

Within the medical domain, ontologies are a fundamental part of computational reasoning, particularly in the realms of precision medicine and explainable AI [8]. Biomedical and health sciences extensively leverage ontologies to encapsulate a vast spectrum of knowledge, spanning diverse realms encompassing diseases [17], gene products [1], phenotypic abnormalities [10], clinical trials [18], vaccine information [11], and human anatomy [14].

Sub-symbolic text models Recent advancements in the realm of natural language processing have witnessed a rise in the popularity of Large Language Models (LLMs). These models, powered by transformer-based neural network architectures, boast an impressive capacity to manipulate, summarize, generate, and predict textual content similar to human language [22]. Leveraging vast text corpora for training, often comprising hundreds of billions of parameters, LLMs excel in content generation within the domain of generative AI.

However, alongside their remarkable capabilities, LLMs have some fundamental limitations. They are prone to misinterpreting instructions, generating biased content and factually incorrect information [21]. These drawbacks highlight a lack of control over the accuracy and consistency of the text generated, leading to concerns such as the proliferation of fake news and instances of plagiarism [9]. Such challenges align with the characterization of LLMs as black box systems, as mentioned by the EU AI Act, where comprehending and interpreting their internal mechanisms pose inherent difficulties.

Symbolic text models The exploration of Logic Programming techniques, particularly utilizing Prolog, Answer Set Programming (ASP), and Constraint Logic Programming (CLP), for extracting concepts from raw data in alignment with xAI standards and subsequently translating them into natural language expressions, remains relatively underexplored in the literature. For instance, [15] proposed models of grammars and graph-based structures leveraging Prolog unification.

Conversely, there has been significant attention from the Logic Programming community towards the problem of processing natural language text as input [16] with Definite Clause Grammars (DCGs) playing a prominent role. DCGs, introduced in the 1980s, serve as a powerful tool for parsing both natural and artificial languages using explicit grammar rules and Prolog [12]. Notably, DCGs offer bidirectional capabilities, enabling not just parsing but also text generation from a controlled context-free grammar conforming to Backus-Naur Form rules. Additionally, they facilitate the modeling of multiple variants of the grammar tree. While examples of formalization of language structure exist in

```
[class(student),
  [rel(attribute),attribute(plural)],
  [rel(attributive_spec),class(course)]]
```

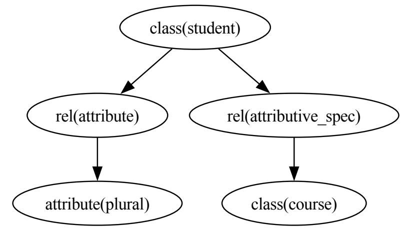


Figure 1: Example of an input concept (the Prolog list and the corresponding tree).

the literature, they are not necessarily rule-based or operational [6]. However, they still provide valuable insights into understanding and modeling language structures.

3 Model

Overall system The system is composed of two modules: Data2Concept and a Concept2Text. The first one is able to identify concepts that are represented as trees of classes and relations among them. The second module can be fed by the first one, but in general it can process any tree concept modelization. The system can therefore describe either specific raw data or general concepts. We refer interested readers to our previous work [4, 5] for further details on Data2Concept.

Concepts We now focus on the design choices of the Concept2Text module. As formalized by ontologies, a general concept can be modeled by a graph where nodes are classes and (hyper-)edges are relations among them. A narration of the graph would talk about nodes (described by their names in the selected language) and would link them according to expressed relations (e.g., verbal and propositional relations). Since the graph translation into a sentence is not straightforward, let us point out a feasible sequence of manipulations that allows us to reach the goal.

At the beginning, a general graph can not be directly translated into a narration, since nodes must be sorted into a narrative order. Moreover, not every edge can be described (especially if common knowledge is present), which requires to filter out many edges. A possible automation would require to control the summary level and the semantic information loss. In the future we plan to investigate this aspect. From now on, we assume to work with a simple spanning tree of the original (sub)graph.

In our experimentation, trees proved to be a consistent data structure that serves the purpose of hosting concept information as well as the various translations towards a corresponding well-formed natural language expression. Therefore, let us illustrate a minimal example about our tree representation, in the case of the concept of *students of a course*. We define two classes *student* and *course* by using a predicate `class/1`, and relate them by means of `rel/1` predicate. For example, we can specify that the class *student* has the attribute *plural* and specify that those students belong to a *course*. Syntactically, we structure a Prolog nested list `[Root,Child`$_1$`,...,Child`$_n$`]`, where children may contain further nested lists. Figure 1 illustrates the encoding and the corresponding tree representation.

Rewriting of trees We argue that a uniform rule-based rewriting system to drive the Concept2Text process is general and modular. During rewriting, trees undergo structural (semantic) and node (syntax) modifications. The rewriting system must be stratified, with the idea to run a set of rewriting rules related to a specific stage until fixpoint, before starting with the next stage. This choice helps in controlling the semantics of changes and well suits for differentiating behaviours that are language dependent.

In a preliminary model we considered using DCGs, but we encountered two main limitations: DCG rules are designed to model a space of trees according to a context free grammar, rather than to control the rewriting process (when a subtree is rewritten into a new subtree). Even if adaptable for tree rewriting, the head of DCG rules does not support a general tree shape. Secondly, we need to control how alternative and equivalent rewriting options are selected, as opposed to allowing Prolog SLD resolution to explore all possibilities.

Let us explore how tree rewriting is conceptualized (see also Section 3.1). When making changes to a tree structure, we need to pinpoint specific conditions that trigger these modifications, typically based on the presence of certain subtrees and their relationships embedded in the larger structure. We expect a rule to encompass the lowest node that is an ancestor of all relevant subtrees, including locations that are affected by the rule rewriting (potentially elsewhere in the tree). Each rule is responsible for constructing a new subtree that replaces the previous content hanging from that node. While some cases involve straightforward substitutions of one subtree with another, more complex scenarios can entail assembling intricate structures by combining existing components, rearranging their structure, and introducing new elements. This level of generality enables the modeling of typical semantic equivalent rewritings as well as grammatical transformations (e.g., active vs. passive voice, word to pronoun substitution, etc.).

The Concept2Text rewriting process can be broken down into several stages, each addressing specific objectives. The overall construction of the final tree benefits from the iterative tree rewriting performed at each stage's fix points. Although the fixed-point rewriting mechanism is common across all stages, we prefer to tailor rules and introduce barriers to fix points. This approach offers several advantages, including the ability to accommodate language-independent stages alongside those requiring language-specific rules. We can also avoid to determine the complete BNF grammar of a natural language, since we can handle each stage separately. The transition from a stage to the next one requires to model rewriting rules rather than complete grammars.

We report on each stage and their purposes:

1. **Equivalent Concepts:** This stage rewrites classes and relations into equivalent semantic versions. The goal is to generate more specific relations that can be devised by common knowledge and to find alternative patterns to express the same meaning. The output remains a concept-based tree. This stage ensures semantic-preserving variants of the concept, leading to the furthest but still equivalent final text.

2. **Concept2Structure:** This stage transforms the tree of concepts and their relations into a prototype of grammar structure. It is mainly a tree shape rewriting with the addition of internal nodes that host future grammar information. It constructs the components of a sentence (noun and verbal subtrees, as well as complements). Classes and relations still retain their ontology descriptions.

3. **Structure2Grammar:** While maintaining the overall structure, this stage translates each class and relation into grammar lexemes and/or other simple grammatical forms.

4. **Coordination:** This stage ensures that subtrees are coordinated, as necessary, to match gender and number for nouns, verbs, etc.

5. **Inflection and Sorting:** Responsible for producing the correct inflections for nouns and adjectives, as well as conjugations for verbs. Also, it computes the correct word order for words within the same phrase through the resolution of language-dependent Constraint Satisfaction Problem (CSP).

6. **Syntax:** Applies local rules to consecutive words to ensure syntactic properties are met (e.g., contractions, ellipsis, etc.).

3.1 Implementation

Our objective is to devise rules flexible enough to handle common language properties, such as subtree swapping and restructuring. The sequence of transformations outlined in the preceding section has been realized using Prolog. Here, we provide a brief overview of the main components of this implementation.

3.1.1 Tree rewriting

Trees are represented as lists of the form `[RootInfo|Children]`, where `Children` is the list of child trees. Each phase of the pipeline takes a tree as input and produces a list of trees as output (representing possible variants according to a rule) obtained by applying a set of rewriting rules. These rule sets are unique and tailored to the specific requirements of each phase. However, all rules are uniformly described by Prolog clauses defining the predicate

```
rule(Lang,Type,Name,Tree,RewTree)
```

where `Lang` specifies the target language of the translation, which remains consistent throughout the process (currently the possible choices are English and Italian); `Type` is the specific phase of the rewriting process (i.e., `equiv_concept`, `concept2structure`, `structure2grammar`, `coordination`, `inflection`, and `syntax`); `Name` is a unique rule ID, distinguishing a specific rewriting among those possible in the phase `Type`; `Tree` and `RewTree` are the input tree and the rewritten tree, resp. In each phase a BFS traversal of the input tree drives rule application and is repeated until a fixpoint is reached (i.e., no more rules of that phase are applicable). While rewriting `Tree`, an auxiliary tree `RuleTree` is also produced. `RuleTree` is isomorphic in shape to `RewTree` and describes the applied rule(s) for each node of `RewTree`.

It is important to note that the information gathered in `RewTree` plays a vital role in ensuring the explainability of the approach. `RewTree` serves as a description of the justification for each rewrite performed. This is achieved by recording the `Name` argument found in the definition of the clause(s) of `rule/5` used in the rewriting. Currently, this information comprises rule IDs, but richer knowledge can be easily managed if needed.

In the final stage (syntax rewriting) the tree is flattened, and leaves are extracted to form a straightforward list of words comprising the sentence. This list of words is then rewritten, until a fixpoint is reached, using a set of rules that only inspect pairs of consecutive words.

Here are some additional details about `rule/5`. Each rule generates a list of trees as alternative variants. When applying a rule, we must select one variant from this list. We have chosen a random selection strategy that takes into account previous choices. This approach has proven particularly effective in preventing repetitions of the same structure in different parts of the final sentence. We keep track of the choice history for each rule using simple assertions. In cases where the same rule is fired multiple times, we ensure that the last choice is avoided, if possible.

3.1.2 Language independent stages

The first stage (Equivalent Concepts), responsible for handling equivalent concepts, is language-independent. While classes and relations must be named according to a specific language (English in the paper), this naming convention does not affect the generation of a specific language, as names will be converted later according to language-dependent rules.

Now, let us introduce a working example to illustrate some key features contained in the stages described above. We model the concept of an *interval* that specifies the use of the class *year* as its unit of measure (*uom*):

```
[class(interval),[rel(attribute),attribute(uom(class(year)))],...]
```

If a common knowledge ontology is accessible, we could discover that `is_a(year,time)` and that the class *interval* can be further specified as an interval of time. Also, additional semantic knowledge about equivalences could inform the rewriting rules that an interval of time is equivalent to the class *period*.

Implementing rules that trigger whenever common knowledge adds some information is straightforward. In this case, a sequence of rewritings could be:

```
[class(interval),[rel(attributive_spec),attribute(class(time))],...]
[class(period),...]
```

where `attributive_spec` represents the specification proposition relation attached to the class *time*.

Let also discuss some potential equivalences that can be drawn for an interval that deals with a range of numbers V_1 and V_2, e.g.:

```
[class(interval),[rel(attribute),attribute(range(V1,V2))],...]
```

Focusing on the treatment of the class *interval*, we can propose various alternative versions. These include: (i) presenting the interval as a simple measure of a range (between ...), (ii) use the interval class (e.g., the interval ...), and (iii) employing a more refined version with the addition of a relative subordinate (e.g., the interval that spans ...). Furthermore, the actual measure itself (the range between two numbers) can be expressed using different prepositions (from ... to, between ..., starting from ... up to ...). The combination of rules that rewrite different parts of the concept, sometimes even depending on one another, generates a combinatorial explosion and produces a rich set of alternative variants already at the concept stage.

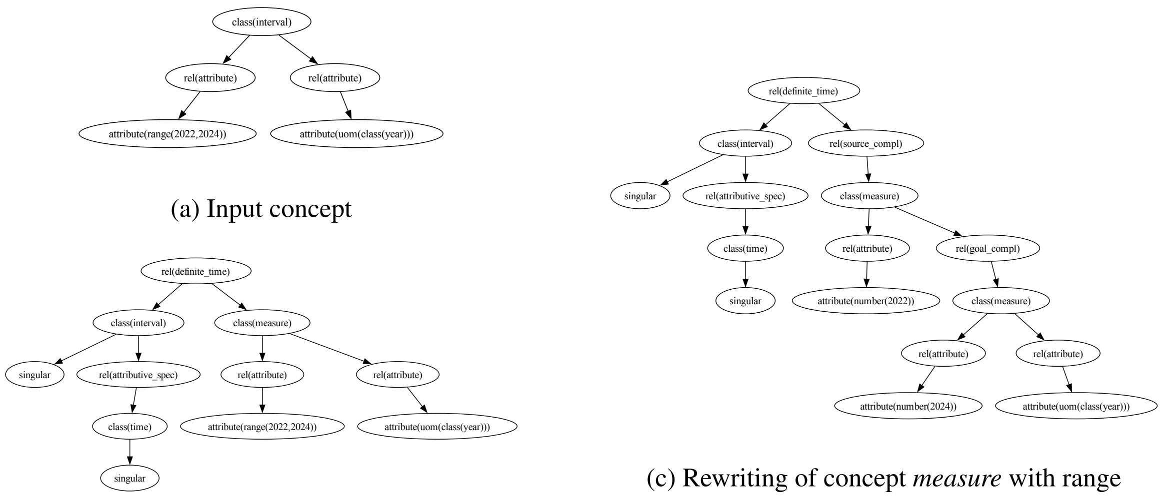
Figure 2 illustrates a simplified example demonstrating the application of the two rewriting rules described above. It is noteworthy how the classes are matched and rewritten: from (a) to (b), the class *interval* is replaced with the left subtree, introducing the concept of measure; from (b) to (c), the class *measure*, denoting a range, is rewritten into a nesting of two complements (source and goal) with measures of simple numeric quantities.

Let us show a simplified snippet of the first rule:

```
rule(_Lang,equiv_concept,equiv_interval, [Root|C],
      [[Root|C2], ... ] ]):- % list of equivalent trees
  % firing condition
  member([class(interval)|C1],C),
  member([rel(attribute),attribute(range(V1,V2))],C1)
  member_non_var([rel(attribute),attribute(uom(class(Uom)))],C1),
  % prepare new tree structure (depending whether isa relation is known)
  ( common_knowledge_isa(Uom,Isa),!,
    Int=[class(interval),[rel(attribute),attribute(singular)],
    [rel(attributive_spec),[class(Isa),singular]]];
    Int=[class(interval),[rel(attribute),attribute(singular)]] ),
  replace(C,[class(interval)|C1],[[rel(definite_time),
     Int,[class(measure)|C1]]],C2),
  ...
```

where we assume defined a predicate `replace/4` that takes the input list, the element to be replaced, the replacement list (enclosed by an additional list, in case multiple elements need to be inserted), and returns the output list.

For the second rule applied in the example, a simplified code snippet could be:



(a) Input concept

(b) Rewriting of concept *interval*

(c) Rewriting of concept *measure* with range

Figure 2: Example of rewriting with concept equivalence.

```
rule(_Lang,equiv_class,measure_range, [Root|C],
      [[Root|C4], ... ]):-  % list of equivalent trees
  member([class(measure)|C1],C),
  member([rel(attribute),attribute(range(N1,N2))],C1),
  ( El=[rel(attribute),attribute(uom(class(U)))],
    member(El,C1),!,Uom=[El];  % there is a UoM specified
    Uom=[] ),
  replace(C,[rel(attribute),attribute(range(N1,N2))],[[]],C2),
  replace(C2,El,[[]],C3),
  % replace subtree at measure class with single measures
  replace(C3,[class(measure)|C1],
          [[rel(source_compl),
            [class(measure),[rel(attribute),attribute(number(N1))]],
             [rel(goal_compl),[class(measure),
              [rel(attribute),attribute(number(N2))]|Uom]]|Uom]]],C4),
   ...
```

The second stage (Concept2Structure) is essentially language independent. Concepts are structurally rearranged into subtrees that model grammar phrases. For instance, a class (future subject) may have a verb relation as one child, and an object may be associated with the verb as its child. This three node branch at the concept level is flattened, resulting in three ordered siblings of phrases (subject, verb, object).

It is also necessary to tag each subtree with information about their phrasal role: e.g. there are nouns, verbs, propositions and relative phrases. The rewriting is able to classify them according to relations and deductions on them. Internal nodes are thus enriched with explicit descriptions of their subtrees, using a predicate `info/4` that specifies the type of phrase (using standard linguistic terminology such as np, vp, pp, rp standing for noun, verbal, propositional, relative phrases), the subtype (e.g., subject, object), and the gender and number attributes that apply to the subtree.

The fourth stage (Coordination) is language independent as it is responsible for structural matching of the gender and/or number variables contained in the `info/4` nodes. Some matches are enforced by default (e.g., between subject and verb), but in other cases, a previous rewriting stage may have forwarded a request for an explicit coordination. For example, when creating a relative subordinate subtree, the phrase must match the gender and number to the antecedent noun. However, there is no guarantee that the noun has already been processed and the associated `info/4` is already created. Only when that stage is over, the variables for gender and number are available. The Coordination stage can then safely find and match the correct variables, upon a request that is embedded as a service node in the subordinate tree. Coordination is enforced via unification of variables.

3.1.3 Language dependent stages

The overall structure allows us to focus on language-dependent rules for specific stages: Structure2Grammar, Inflection, and Syntax. One advantage of this model is that we can easily plug in sets of rules without modifying the system.

The Structure2Grammar stage translates classes and relations into their corresponding grammar lexemes (the non-inflected roots of words) in the target language. A general application would require a complete association of synonyms for each class. While we are exploring automated tools to streamline this phase, for small domain-specific applications, manual crafting is a viable option. Synonyms are modeled as a list of lexemes in the variants of rules. The tree nodes can be replaced accordingly. Each node is encapsulated by a parent node that provides its type (e.g., noun, adjective, number, verb, preposition, etc.). This tagging allows a simple reasoning when determining the correct order of elements within a phrase. In the future we plan to find a more accurate and automated model for the choice of appropriate lexemes, based on context and common knowledge. A fluent sentence can greatly benefit from this choice.

Let us now provide some details about the inflection stage. Here, inflections for nouns, adjectives, and verbs are selected based on their gender, number and verbal tense. This is typically accomplished by consulting a dictionary that explicitly associates lexemes with words. Auxiliary verbs are generated according to the rules of the target language.

The inflection stage also arranges the words in the correct order within each phrase. Languages have various rules governing the order of words in a phrase, and attempting to cover all possible cases would be impractical due to the combinatorial explosion of possibilities. To address this challenge, we have devised a set of rules that describe local and partial orderings among subsets of words within the phrase. By combining these partial orders, we can derive the correct total order of words. These orderings may depend on word types and specific words themselves.

We represent this network of sorting constraints as a Constraint Satisfaction Problem (CSP). While solving the CSP itself is straightforward, developing accurate order constraints requires careful tuning. Thus, the flexibility of a CSP allows to support the updates of the constraints considered.

The final stage (Syntax) addresses the enforcement of writing rules for adjacent words. Every language possesses distinct rules governing word combinations, typically controlled through local pattern matching at the word or character level. For instance, this stage handles scenarios where two words are merged into contractions or a single letter is removed/added (e.g. *a increase* is converted to *an increase* because of the presence of a vowel at the beginning of the second word). This stage considers the leaves of the tree output by the Inflection stage. If read according to a Depth-First Search (DFS) traversal, the list of words compose the final sentence. Internal nodes describe structural properties of sub-trees. At this stage, punctuation marks and correct spacing are handled. Moreover, internal nodes can also be exploited in case the sentence requires some markup. We experimented with HTML tagging, which can

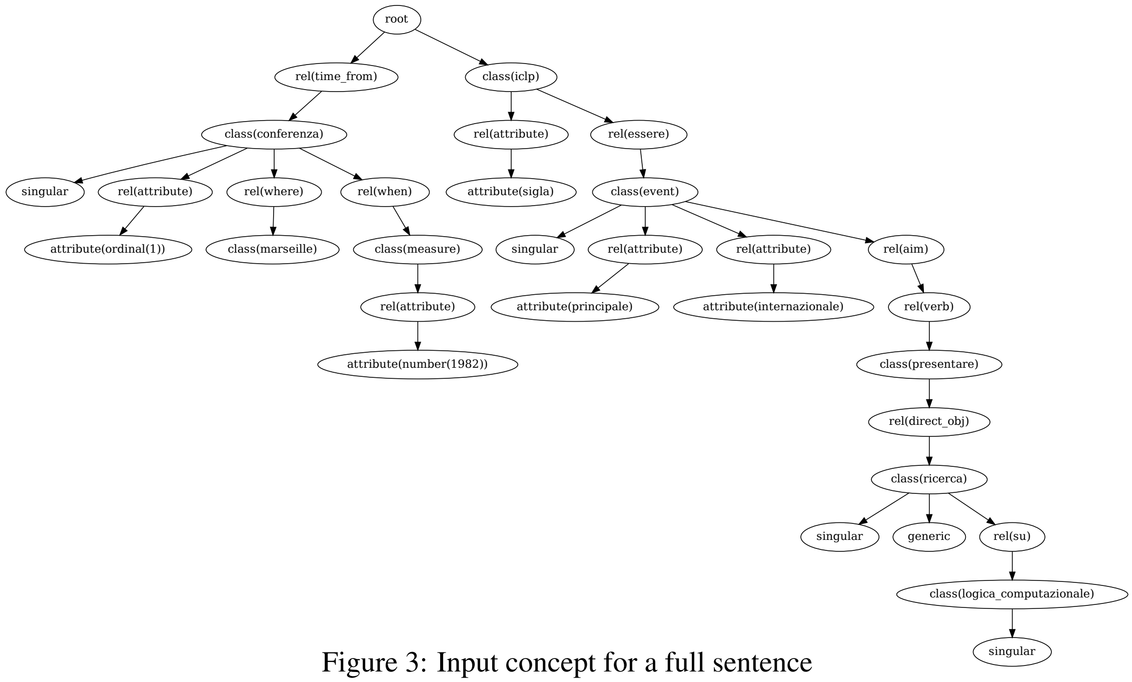


Figure 3: Input concept for a full sentence

be easily incorporated into rules.

4 Results

General concept narration We first test the potential and robustness of the Concept2Text system only. We crafted a concept that is depicted in Figure 3. The concept describes the traditional first sentence appearing in the call for papers of ICLP2024. It can be noted that we supported generic relations of the kind *where* and *when* associated to the class *conference*. Semantic equivalences can handle variants that involve relative subordinates, different priority in concept order and active/passive forms that greatly influence the next stages, independently on the selected language. This showcases how complex handling of relations can be performed and suggests how the underlying machinery can be adapted to many other cases, by simply changing classes and relations taken from this prototype concept. In the supplementary material we show a sequence of tree rewritings after reaching the fixed point of each stage, starting from the input of Figure 3.

We locally checked each rewriting rule behaviour, along with the variants produced. This process is rather convenient while visually inspecting the rewritten trees. Clearly, the complete interaction among rules and stages grows exponentially and full checks can be performed on a single specific trace of the program.

For the English language specialized rules set, we (non-exhaustively) collected more than 13,000 unique sentences, while for Italian we counted 3,200+ sentences. Interested readers can download and consult the list of sentences at `ahead-lab.unipr.it/files-for-iclp2024/`. We manually reviewed some samples and they all appear well-formed. One advantage of the rule-based model is that errors could be quickly debugged and fixed while developing the system. Even if the variants for rules are rather limited in number (at most 4 versions per rule) and certainly they will be expanded in the future, this result shows how the combinatorics of various stages allows to enumerate a surprisingly high set of semantics preserved sentences.

It is noteworthy how the rewriting of equivalence and grammar structures is capable of providing remarkable differences, while preserving semantics perfectly, thanks to the strict transitivity of the applied equivalences. The resulting text appears natural, although some additional synonyms could be included to enrich and diversify certain words.

We also tested some free LLMs available online (i.e., ChatGPT 3.5, Gemini). We asked to generate 10 instances of a sentence that strictly preserved the semantics intended by the Prolog list as in Figure 3. Even if in general the results were rather accurate, in some cases some attributes were skipped and/or some verb choices were not perfectly compatible with the context. It was also difficult to force the complete adherence with the original input via prompting. Those are a set of secondary issues, since the tested methodology does not comply with the major requirement of explainability.

Comparing the control obtained from a well-crafted prompt to the LLM, our pipeline ensures that the overall semantic integrity is maintained consistently from input to the final sentence throughout each step of rewriting. In contrast, LLMs may introduce arbitrary choices or hallucinations, affecting both semantic and syntactic levels. Additionally, we observed that it is challenging to enforce the use of all provided attributes, as LLMs tend to interpret the intended semantics with limited control over the degree of summarization and the relative importance of each attribute.

Data2Concept narration Our Data2Concept system (see introduction) analyzes data series and outputs concepts in the form of trees that contain data properties as well as confidence about accuracy and relevance of the findings. The Concept2Text system can be attached and the full pipeline can be tested. We report here on a simple data analysis run on the number of papers about explainable AI indexed by Scopus each year from 2014 to 2023. The data series (y values) is $[0, 2, 0, 7, 84, 217, 428, 816, 1266, 1905]$ and the contextual classes are about the x axis (year), the x axis list of values (numbers between 2014 and 2023), the y axis class (papers) and the overall class of the data series (publications on xAI). The accuracy of the extracted concepts are computed on a integer value on the range $0 \ldots 100$, and they are mapped into a judgment scale of adjectives. Table 1 shows two sentences produced by the pipeline for both English and Italian. We can note that the adherence to the original series is very high, as stated by the accuracy feedback.

5 Conclusions

The paper presented an explainable methodology to rewrite a concept in well-formed natural languages. The system adopts a multi-level rewriting procedure that can produce semantic, grammar and lexical variants that are aware of the context. Common knowledge can be used to better adapt to specific contexts. The system is modular, since it allows for easy adaptation to various domain-specific applications and output languages. Moreover, the system adheres to explainable AI standards by offering transparency and verifiability. We can conclude that the Data2Concept2Text complete system is effective in modeling general concepts and to translate them into multiple languages with the same architecture. As applications, it can handle both raw data series and narration of general concepts from ontologies.

This work opens different lines of research to be further explored. Adapting different rules for handling grammar and syntax from different languages requires some human time, since this kind of formalization is often fuzzy and requires language experts. Developing a comprehensive rule set for accurately translating classes into suitable grammar synonyms is a complex task, as the most appropriate choices

Table 1: Output examples. 1–2 for English and 3–4 for Italian.

1. From the year 2014 up to 2023 publications in explainable AI have exponentially grown in an important way (from 0 up to 1905) [excellent accuracy]; in detail, during the interval of time between the years 2014 and 2017 publications have been significantly steady (from 0 to 7) [excellent accuracy].
2. There has been an important exponential growth of publications on explainable AI (from 0 up to 1905) during the interval of time that has spanned starting from the year 2014 up to 2023 [excellent accuracy]; specifically, between the years 2014 and 2017 publications have shown themselves to be significantly constant (from 0 up to 7) [excellent accuracy].
3. C'è stato un incremento esponenziale importante di pubblicazioni sulla IA spiegabile (da 0 a 1905) dagli anni 2014 ai 2023 [accuratezza ottima]; in dettaglio, dall'anno 2014 e durante i 3 anni successivi le pubblicazioni sono state decisamente stabili (da 0 fino a 7) [accuratezza ottima].
4. Nell'intervallo di tempo dagli anni 2014 ai 2023 le pubblicazioni sulla IA spiegabile sono aumentate esponenzialmente in modo importante (a partire da 0 fino a 1905) [accuratezza ottima]; in dettaglio, nel periodo dall'anno 2014 fino al 2017 i lavori sono stati decisamente costanti (da 0 a 7) [accuratezza ottima].

heavily depend on context. We aim to devise automatic methods to retrieve such preferences and stylistic usages.

Variants of rules can be classified according to verbosity and style. This information can help to match preferences about properties of the final text to be produced.

We also plan to investigate the handling of general concept graphs, rather than our tree-like set of relations on concepts. Converting this graph into a spanning tree, or alternatively, synthesizing the information in a controlled manner, could facilitate the creation of guided concept summaries.

The entire pipeline is versatile and applicable across various domains, particularly in scenarios where reports are generated based on data analytics. We intend to extend this methodology to automated analysis of ECGs and other medical data, financial data, and more broadly, to produce trustworthy Business Intelligence (explainable automated reporting).

References

[1] Michael Ashburner et al. (2000): *Gene ontology: tool for the unification of biology*. *Nature genetics* 25(1), pp. 25–29, doi:10.1038/75556.

[2] Alejandro Barredo Arrieta et al. (2020): *Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI*. *Inf. Fusion* 58, pp. 82–115, doi:10.1016/j.inffus.2019.12.012.

[3] Flavio Bertini, Alessandro Dal Palù, Andrea Formisano, Andrea Pintus, Sara Rainieri & Luana Salvarani (2023): *Students' Careers and AI: a decision-making support system for Academia*. In: *Proceedings of the Italia Intelligenza Artificiale (Ital-IA 2023), Pisa, Italy, May 29-30, 2023*, *CEUR Workshop Proceedings* 3486, CEUR-WS.org, pp. 272–277. Available at `https://ceur-ws.org/Vol-3486/`.

[4] Alessandro Dal Palù, Agostino Dovier & Andrea Formisano (2023): *Towards Explainable Data-to-text Generation*. In Agostino Dovier & Andrea Formisano, editors: *Proceedings of the 38th Italian Conference on Computational Logic, Udine, Italy, June 21-23, 2023*, *CEUR Workshop Proceedings* 3428, CEUR-WS.org, pp. 1–15. Available at `https://ceur-ws.org/Vol-3428/`.

[5] Alessandro Dal Palù, Agostino Dovier & Andrea Formisano (2023): *An xAI Approach for Data-to-Text Processing with ASP*. In Enrico Pontelli et al., editors: *Proc. 39th International Conference on Logic Programming, ICLP 2023, Imperial College London, UK, 9th July 2023 - 15th July 2023*, *EPTCS* 385, pp. 353–366, doi:10.4204/EPTCS.385.38.

[6] Marcel den Dikken (2013): *The Cambridge handbook of generative syntax*. Cambridge Univ. Press, doi:10.1017/CBO9780511804571.

[7] European Commission (2021): *Regulation of the European Parliament and of the Council. Laying Down Harmonised Rules on Artificial Intelligence (Artificial Intelligence Act)*. `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:52021PC0206`.

[8] Melissa A Haendel, Christopher G Chute & Peter N Robinson (2018): *Classification, ontology, and precision medicine*. *New England Journal of Medicine* 379(15), pp. 1452–1462, doi:10.1056/NEJMra1615014.

[9] Ziwei Ji et al. (2023): *Survey of hallucination in natural language generation*. *ACM Computing Surveys* 55(12), pp. 1–38, doi:10.48550/ARXIV.2405.20315.

[10] Sebastian Köhler et al. (2021): *The human phenotype ontology in 2021*. *Nucleic acids research* 49(D1), pp. D1207–D1217, doi:10.1093/NAR/GKAA1043.

[11] Yu Lin & Yongqun He (2012): *Ontology representation and analysis of vaccine formulation and administration and their effects on vaccine immune responses*. *J. Biomed. Semant.* 3, pp. 1–15, doi:10.1186/2041-1480-3-17.

[12] Yuji Matsumoto, Hozumi Tanaka, Hideki Hirakawa, Hideo Miyoshi & Hideki Yasukawa (1983): *BUP: a bottom-up parser embedded in Prolog*. *New Generation Computing* 1(2), pp. 145–158, doi:10.1007/BF03037421.

[13] Deborah L. McGuinness, Frank Van Harmelen et al. (Updated in 2009): *OWL web ontology language overview*. *W3C recommendation 10 February 2004*. Available at `www.w3.org/TR/owl-features/`.

[14] Natalya F. Noy, Mark A. Musen, José L.V. Mejino Jr & Cornelius Rosse (2004): *Pushing the envelope: challenges in a frame-based representation of human anatomy*. *Data Knowl. Eng.* 48(3), pp. 335–359, doi:10.1016/J.DATAK.2003.06.002.

[15] Fernando C. N. Pereira (1987): *Grammars and Logics of Partial Information*. In Jean-Louis Lassez, editor: *Logic Programming, Proceedings of the Fourth International Conference, Melbourne, Victoria, Australia, May 25-29, 1987*, MIT Press, pp. 989–1013. ISBN:0262121255.

[16] Fernando C. N. Pereira & Stuart M. Shieber (2002): *Prolog and natural-language analysis*. Microtome Publishing. ISBN:0937073180.

[17] Lynn M. Schriml et al. (2019): *Human Disease Ontology 2018 update: classification, content and workflow expansion*. *Nucleic acids research* 47(D1), pp. D955–D962, doi:10.1093/NAR/GKY1032.

[18] Barry Smith et al. (2007): *The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration*. *Nature biotechnology* 25(11), pp. 1251–1255, doi:10.1038/nbt1346.

[19] Robyn Speer, Joshua Chin & Catherine Havasi (2017): *Conceptnet 5.5: An open multilingual graph of general knowledge*. In: *Proceedings of the 31st AAAI conference on artificial intelligence*, pp. 4444–4451, doi:10.1609/AAAI.V31I1.11164.

[20] Steffen Staab & Rudi Studer, editors (2009): *Handbook on Ontologies*. International Handbooks on Information Systems, Springer, doi:10.1007/978-3-540-92673-3.

[21] 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 arXiv:2307.12966*, doi:10.48550/ARXIV.2307.12966.

[22] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong et al. (2023): *A survey of large language models*. *arXiv preprint arXiv:2303.18223*, doi:10.48550/ARXIV.2303.18223.

Appendix

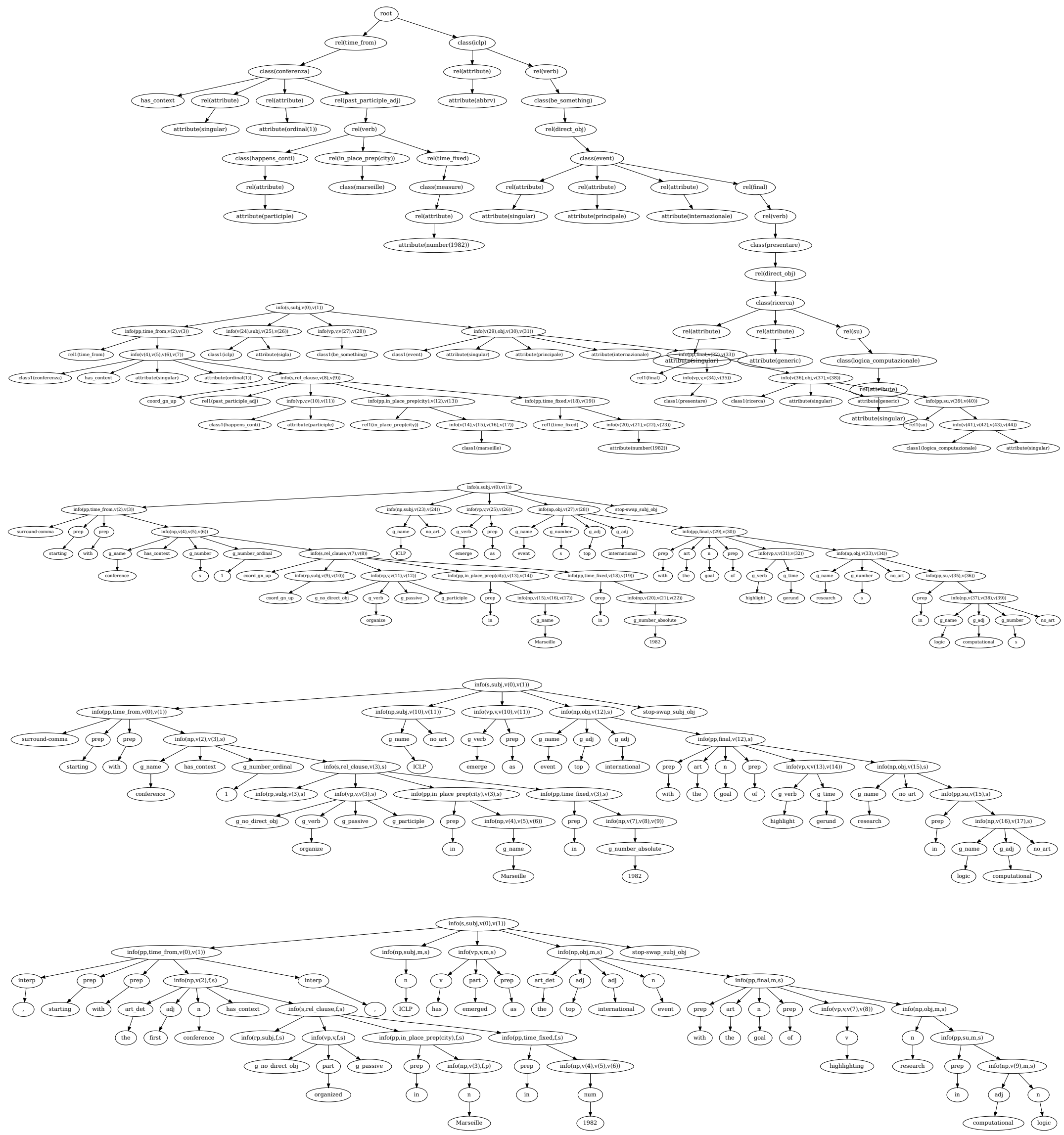


Figure 4: Computed stages for the call for papers concept of Figure 3. From top to bottom: Concept Equivalence, Concept2Structure, Structure2Grammar, Coordination, Inflection

Counterfactual Explanations as Plans

Vaishak Belle
University of Edinburgh & Alan Turing Institute, UK
vaishak@ed.ac.uk

There has been considerable recent interest in explainability in AI, especially with black-box machine learning models. As correctly observed by the planning community, when the application at hand is not a single-shot decision or prediction, but a sequence of actions that depend on observations, a richer notion of explanations are desirable.

In this paper, we look to provide a formal account of "counterfactual explanations," based in terms of action sequences. We then show that this naturally leads to an account of model reconciliation, which might take the form of the user correcting the agent's model, or suggesting actions to the agent's plan. For this, we will need to articulate what is true versus what is known, and we appeal to a modal fragment of the situation calculus to formalise these intuitions. We consider various settings: the agent knowing partial truths, weakened truths and having false beliefs, and show that our definitions easily generalize to these different settings.

1 Introduction

There has been considerable recent interest in explainability in AI, especially with black-box machine learning models, given applications in credit-risk analysis, insurance pricing and self-driving cars. Much of this focus is on single-shot decision or prediction, but as correctly observed by the automated planning community [14, 7, 6], in applications involving sequence of actions and observations and conditional plans that depend on observations, a richer notion of explanations are desirable. Fox et al. [14], for instance, argue that understanding why a certain action was chosen (and not some other), why one sequence is more optimal than another, etc, are all desired constructs in explanations.

Despite all this attention, there is yet to emerge a theory on how exactly to frame explanations in a general way. One candidate is the notable line of work on model reconciliation [40]. The idea is that the agent might have incomplete or even false information about the world, and the user can advise the agent by correcting the agent's model, or suggesting actions to the agent's plan. The latter move is in the spirit of human-aware AI [19]. But such an idea is inherently *epistemic*, and this brings the explainable AI literature closer to *epistemic planning* [2]. Many recent threads of work have tried to explicate this connection.

In [37], for example, the idea of discrepancy and resolving such discrepancy is studied. Using the epistemic logic fragment from [30], where so-called proper epistemic knowledge bases are chosen for initial plan states that allow only for modal literals, discrepancy emerges when (in a dynamic logic-like language) $\Sigma \models [\delta]\boldsymbol{K}_i\phi$ but $\Sigma \not\models [\delta]\boldsymbol{K}_j\phi$. Given a goal ϕ, an action sequence δ, background knowledge Σ for agents i and j which include dynamic axioms, it turns out that i believes that ϕ is true but j does not. So the resolution is to find some course of action δ' that ensures that either $\Sigma \models [\delta'](\boldsymbol{K}_i\phi \land \boldsymbol{K}_j\phi)$ or that $\Sigma \models [\delta'](\boldsymbol{K}_i\neg\phi \land \boldsymbol{K}_j\neg\phi)$. That is, they both jointly believe after δ' that ϕ is made true, or that ϕ is made false. And as pointed out in [37], articulating the difference between beliefs and ground truth is needed for clarity.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 153–167, doi:10.4204/EPTCS.416.14

Likewise, the works on *contrastive explanations* [21] as well as the credulous/skeptical semantics for model reconciliation from [42] are related. In the former, the "why" question is tackled by offering to add or remove actions from the current plan. In the latter, a notion of credulous and skeptical entailment is suggested as a way to keep updated mental states consistent in service of explanations. Credulous entailment is when a single belief state suffices to check the validity and achievement of a plan, and skeptical entailment when every belief state is involved. The problem of "explanation generation", then, between the user's theory Σ and the agent's theory Σ' is triggered when $\Sigma \models [\delta]\phi$ for some goal ϕ after sequence δ but $\Sigma' \not\models [\delta]\phi$. This necessitates the updating of Σ' to Σ'' such that $\Sigma'' \models [\delta]\phi$. Note that, even though skeptical/credulous entailment involves belief states, the formalism itself is not epistemic. Perhaps it does not need to be in some limited cases, but ultimately a distinction between truth in the real-world and beliefs provides clarity on how the agent's model needs to be adjusted when plans do not work, either because its missing actions or entertaining incomplete/false truths.

The reader may also surmise that there is clearly some relationship between these proposals, but it is not spelt out. In fact, no clear logical justification is given as why the definitions are reasonable in the first place. As we mentioned, a theory on how exactly to frame explanations in such epistemic settings is yet to emerge.

What we seek to do in this paper is to develop and formalize "counterfactual explanations" over plans in the presence of both physical and sensing actions. Counterfactual explanations in machine learning are widely popular [43], because they provide an intuitive account of "what if" and "what could have been", which helps us realize alternative worlds where a desired outcome might be achieved.[1] Our account can be seen as attempting to establish a logical relationship between contrastive explanations and knowledge based updates for model reconciliation and/or discrepancy. In fact, this relationship is a very simple one: it can be seen as a counterfactual! The "why," "what if", and other such "wh"-questions can be interpreted as counterfactuals [31]. In the static setting, we are simply interested in an alterative world where some properties are different. In the dynamic setting, we might also be interested in an alternative plan that achieves a different outcome, very much in the spirit of contrastive plans. To our knowledge, there is no general account in the literature on counterfactual explanations as plans.

This brings us to the choice of the representation language. As already discussed above, we need to identify, in the first instance, a knowledge representation language for articulating things like what is true, what is known, whether something is not known, and whether something is falsely believed. This will give us an opportunity to formalize explanations in an epistemically adequate representation language. Although proposals such as [30, 37] might be perfectly reasonable, and might even be preferred for an automated planning account, but they do come with various syntactic stipulations that might affect the properties of the logic (e.g., disjunctive entailments). We believe the characterization is more easily stated and readability improved by considering a general knowledge representation language, but nothing in the formalisation necessitates one choice of language over another. In fact, the underlying implementation can involve any of the recent proposals from the epistemic planning literature, e.g., [30, 39], of which we think [37] is particularly fitting. Our notion of counterfactual plans could be easily adapted from the algorithm of that work.

Dimensions to formalisation When attempting to formalise counterfactual (CF) explanations, as we shall show, there are multiple dimensions under which a definition can be explored. In the simplest case, given a plan δ that achieves ϕ, a CF explanation might be a plan δ' that negates ϕ. An interpretation could be as follows: if a plan ends up denying the loan to individual b, find a plan that approves the loan

[1] These notions will vary slightly in the dynamic version. We will focus on identifying the course of actions that can toggle the outcome (i.e., "which plan"), and the information that is necessary to enable a property (i.e., "what should be known").

for b, but ensure that it is "close" to the original plan. This roughly captures counterfactuals in ML: given a data point x that has label y, find a data point x' that is minimally distant from x and has label $\neg y$. We can further concretize this by explicating what closeness means, and whether additional constraints can be provided so that a diverse range of x' are found [29].

But when viewed through the lens of an interaction between an automated agent and a human user, in the sense of human-guided planning [19, 40], the framework becomes richer. We will consider the case where the user assists in achieving goals. So this leads to a broader notion of counterfactuals: consider an alternate history or additional knowledge such that the goal now becomes true. This will require us to articulate the difference between what is true in the real world (the user's knowledge) versus what is believed (by the agent). Interestingly, this does not require a modelling language with multiple agents because the user's knowledge can serve as proxy for truth in the real world.[2] We consider various settings: the agent knowing partial truths, weakened truths and having false beliefs, and show that our definitions easily generalize to these different settings. The discrepancy model from [37], the skeptical/credulous entailment model from [42], among others, can be seen as variations of this more general recipe in epistemic logic. We focus on the mathematical aspects here, but as mentioned, implementations from works such as [37] can be adapted for generating CF explanations as plans.

This then brings us to the choice of the formal language for our exposition. We choose the situation calculus [34], which has been well-explored for formalizing the semantics of planning problems [32, 34, 15, 27]. But because we are interested in nested beliefs, we explore a (newer) modal variant, the logic $\mathcal{ES}$ [23], which provides a simpler semantics for planners [8], as well closely mirrors the semantics of proposals such as dynamic logic. Perhaps what makes it most interesting that under some conditions reasoning about actions and knowledge can reduce to non-modal (first-order or propositional) reasoning [23], a feature we feel has not been considered extensively in the planning community. Our ideas do not hinge on this language, and so any planning language that helps us reason about truth, knowledge, actions and sensing should suffice.

2 A Logic for Knowledge and Action

We now introduce the logic $\mathcal{ES}$ [23].[3] It is an epistemic logic, but we only need the objective fragment for formalizing counterfactual explanations as plans. When we consider the more elaborate notion of reconciliation-type explanations where both a user and an agent is necessary, we will use the full language.

The non-modal fragment of $\mathcal{ES}$ consists of standard first-order logic with $=$. That is, connectives $\{\wedge, \forall, \neg\}$, syntactic abbreviations $\{\exists, \equiv, \supset\}$ defined from those connectives, and a supply of variables variables $\{x, y, \ldots, u, v, \ldots\}$. Different to the standard syntax, however, is the inclusion of (countably many) *standard names* (or simply, names) for both objects and actions $\mathcal{R}$, which will allow a simple, substitutional interpretation for $\forall$ and $\exists$. These can be thought of as special extra constants that satisfy

[2]This leads to a single-agent version formulation of, for example, the discrepancy condition from [37] in that it emerges whenever the user/root agent/real world is one where $\Sigma \models [\delta]\phi$ but as far as the agent is concerned: $\Sigma \not\models [\delta]\boldsymbol{K}\phi$. An account with multi-agent modal operators is possible in a straightforward way, using e.g., [36, 20] and [3] in particular, which is a many-agent extension to the language used in this paper.

[3]Our choice of language may seem unusual, but it is worth noting that this language is a modal syntactic variant of the classical epistemic situation that is better geared for reasoning about knowledge [22]. But more importantly, it can be shown that reasoning about actions and knowledge reduces to first-order reasoning via the so-called regression and representation theorems [23]. (For space reasons, we do not discuss such matters further here.) There are, of course, many works explicating the links between the situation calculus and logic programming; see [34] for starters.

the unique name assumption and an infinitary version of domain closure.

Like in the situation calculus, to model immutable properties, we assume rigid predicates and functions, such as $IsPlant(x)$ and $father(x)$ respectively. To model changing properties, $\mathcal{ES}$ includes fluent predicates and functions of every arity, such as $Broken(x)$ and $height(x)$. Note that there is no longer a situation term as an argument in these symbols to distinguish the fluents from the rigids. For example, $\mathcal{ES}$ also includes a distinguished fluent predicates $Poss$ and SF to model the executability of actions and capture sensing outcomes respectively, but they are now a unary predicates. Terms and formulas are constructed as usual. The set of ground atoms $\mathcal{P}$ are obtained by applying all object names in $\mathcal{R}$ to the predicates in the language.

There are four modal operators in $\mathcal{ES}$: $[a], \square, \boldsymbol{K}$ and $\boldsymbol{O}$. For any formula α, we read $[a]\alpha, \square\alpha$ and $\boldsymbol{K}\alpha$ as "α holds after a", "α holds after any sequence of actions" and "α is known," respectively. Moreover, $\boldsymbol{O}\alpha$ is to be read as "α is only-known." Given a sequence $\delta = a_1 \cdots a_k$, we write $[\delta]\alpha$ to mean $[a_1]\cdots[a_k]\alpha$. We write $a \cdot \delta \cdot a'$ to mean $[a] \cdot [a_1] \cdots [a_k] \cdot [a']$.

In classical situation calculus parlance, we would use $[a]\alpha$ to capture successor situations as properties that are true after an action in terms of the current state of affairs. Together with the $\square$ modality, which allows to capture quantification over situations and histories, basic action theories can be defined. Like in the classical approach, one is interested in the entailments of the basic action theory.

Semantics Recall that in the simplest setup of the possible-worlds semantics, worlds mapped propositions to $\{0,1\}$, capturing the (current) state of affairs. $\mathcal{ES}$ is based on the very same idea, but extended to dynamical systems. So, suppose a world maps $\mathcal{P}$ and $\mathcal{Z}$ to $\{0,1\}$.[4] Here, $\mathcal{Z}$ is the set of all finite sequences of action names, including the empty sequence $\langle\rangle$. Let $\mathcal{W}$ be the set of all worlds, and $e \subseteq \mathcal{W}$ be the *epistemic state*. By a *model*, we mean a triple (e,w,z) where $z \in \mathcal{Z}$.

Intuitively, each world can be thought of a situation calculus tree, denoting the properties true initially but also after every sequence of actions. $\mathcal{W}$ is then the set of all such trees. Given a triple (e,w,z), w denotes the real world, and z the actions executed so far. Interestingly, e captures the accessibility relation between worlds, but by modeling the relation as a set, we are enabling positive and negative introspection using a simple technical device.

To account for how knowledge changes after (noise-free) sensing, one defines $w' \sim_z w$, which is to be read as saying "w' and w agree on the sensing for z", as follows:

- if $z = \langle\rangle$, $w' \sim_z w$ for every w'; and
- $w' \sim_{z\cdot a} w$ iff $w' \sim_z w$, and $w'[SF(a),z] = w[SF(a),z]$.

This is saying that initially, we would consider all worlds compatible, but after actions, we would need the world w' to agree on sensing outcomes. The reader might notice that this is clearly a reworking of the successor state axiom for the knowledge fluent in [35].

With this, we get a simply account for truth. We define the satisfaction of formulas wrt the triple (e,w,z), and the semantics is defined inductively:

- $e,w,z \models p$ iff p is an atom and $w[p,z] = 1$;
- $e,w,z \models \alpha \wedge \beta$ iff $e,w,z \models \alpha$ and $e,w,z \models \beta$;
- $e,w,z \models \neg\alpha$ iff $e,w,z \not\models \alpha$;
- $e,w,z \models \forall x\alpha$ iff $e,w,z \models \alpha^x_n$ for all $n \in \mathcal{R}$;

[4] We need to extend the mapping to additionally interpret fluent functions and rigid symbols, omitted here for simplicity.

- $e, w, z \models [a]\alpha$ iff $e, w, z \cdot a \models \alpha$;
- $e, w, z \models \Box\alpha$ iff $e, w, z \cdot z' \models \alpha$ for all $z' \in \mathscr{Z}$;
- $e, w, z \models \boldsymbol{K}\alpha$ iff for all $w' \sim_z w$, if $w' \in e$, $e, w', z \models \alpha$;
- $e, w, z \models \boldsymbol{O}\alpha$ iff for all $w' \sim_z w$, $w' \in e$, iff $e, w', z \models \alpha$.

To define entailment for a logical theory, we write $\Sigma \models \alpha$ (read as "Σ entails α") to mean for every $M = (e, w, \langle\rangle)$, if $M \models \alpha'$ for all $\alpha' \in \Sigma$, then $M \models \alpha$. We write $\models \alpha$ (read as "α is valid") to mean $\{\} \models \alpha$.

Properties Let us first begin by observing that given a model (e, w, z), we do not require $w \in e$. It is easy to show that if we stipulated the inclusion of the real world in the epistemic state, $\boldsymbol{K}\alpha \supset \alpha$ would be true. That is, suppose $\boldsymbol{K}\alpha$. By the definition above, w is surely compatible with itself after any z, and so α must hold at w. Analogously, properties regarding knowledge can be proven with comparatively simpler arguments in a modal framework, in relation to the classical epistemic situation calculus. Valid properties include: (a) $\Box(\boldsymbol{K}(\alpha) \wedge \boldsymbol{K}(\alpha \supset \beta) \supset \boldsymbol{K}(\beta))$; (b) $\Box(\boldsymbol{K}(\alpha) \supset \boldsymbol{K}(\boldsymbol{K}(\alpha)))$; (c) $\Box(\neg\boldsymbol{K}(\alpha) \supset \boldsymbol{K}(\neg\boldsymbol{K}(\alpha)))$; (d) $\Box(\forall x.\ \boldsymbol{K}(\alpha) \supset \boldsymbol{K}(\forall x.\ \alpha))$; and (e) $\Box(\exists x.\ \boldsymbol{K}(\alpha) \supset \boldsymbol{K}(\exists x.\ \alpha))$.

Note that such properties hold over all possible action sequences, which explains the presence of the $\Box$ operator on the outside. The first is about the closure of modus ponens within the epistemic modality. The second and third are on positive and negative introspection. The last two reason about quantification outside the epistemic modality, and what that means in terms of the agent's knowledge. For example, item 5 says that if there is some individual n such that the agent knows $Teacher(n)$, it follows that the agent believes $\exists x Teacher(x)$ to be true. This may seem obvious, but note that the property is really saying that the existence of an individual in some possible world implies that such an individual exists in all accessible worlds. It is because there is a fixed domain of discourse that these properties come out true; they are referred to a the Barcan formula.

It is worth nothing that in single-agent epistemic planning [2], it is most common to have epistemic goals of the sort $\boldsymbol{K}\phi$, $\neg\boldsymbol{K}\phi$ and $\boldsymbol{K}\neg\boldsymbol{K}\phi$, where ϕ is non-modal. The idea is that we might be interested in interleaving physical and sensing actions such that (respectively) ϕ becomes known, or as an observer (e.g., user) we make note that the agent does not know ϕ, or that the agent knows that it does not know ϕ, in which case it might choose to do actions so that it gets to know ϕ. Multiple nestings of modalities are allowed but usually not necessary in the single-agent case. When multiple agents are involved, however, [20, 30, 3], it becomes necessary to interleave epistemic operators, often arbitrarily, in service of notions such as common knowledge [18].

As seen above, the logic $\mathscr{E}\mathscr{S}$ allows for a simple definition of the notion of only-knowing in the presence of actions [26], which allows one to capture both the beliefs as well as the non-beliefs of the agent. Using the modal operator $\boldsymbol{O}$ for only-knowing, it can be shown that $\boldsymbol{O}\alpha \models \boldsymbol{K}\beta$ if $\alpha \models \beta$ but $\boldsymbol{O}\alpha \models \neg\boldsymbol{K}\beta$ if $\alpha \not\models \beta$ for any non-modal $\{\alpha, \beta\}$. That is, only-knowing a knowledge base also means knowing everything entailed by that knowledge base. Conversely, it also means not believing everything that is not entailed by the knowledge base. In that sense, $\boldsymbol{K}$ can be seen as an "at least" epistemic operator, and $\boldsymbol{O}$ captures both at least and "at most" knowing. This can be powerful to ensure, for example, that the agent provably does not know protected attributes.

We will now consider the axiomatization of a basic action theory in $\mathscr{E}\mathscr{S}$. But before explaining how successor state axioms are written, one might wonder whether a successor state axiom for $\boldsymbol{K}$ is needed, as one would for *Knows* in the epistemic situation calculus. It turns out because the compatibility of the worlds already accounted for the executability of actions and sensing outcomes in accessible worlds, such an axiom is actually a property of the logic:

$$\models \Box[a]\boldsymbol{K}(\alpha) \equiv (SF(a) \wedge \boldsymbol{K}(SF(a) \supset [a]\alpha)) \vee (\neg SF(a) \wedge \boldsymbol{K}(\neg SF(a) \supset [a]\alpha)).$$

(Free variables are implicitly quantified from the outside.) What will be known after an action is based on what is true in the real world and the incorporation of this information with the agent's knowledge.

Basic Action Theories To illustrate the language towards the axiomatization of the domain, we consider the analogue of the basic action theory in the situation calculus [34]. It consists of:

- axioms that describe what is true in the initial states, as well as what is known initially;
- precondition axioms that describe the conditions under which actions are executable using a distinguished predicate *Poss*;
- successor state axioms that describe the conditions under which changes happen to fluents on executing actions, incorporating Reiter's monotonic solution to the frame problem; and
- sensing axioms that inform the agent about the world using a distinguished predicate *SF*.

Note that foundational axioms as usually considered in Reiter's variant of the situation calculus [34] are not needed as the tree-like nature of the situations is baked into the semantics.

We will lump the successor state, precondition and sensing axioms as Σ_{dyn}. The sentences that are true initially will be referred to by Σ_0. When we are not interested in epistemic goals, and do not concern ourselves with sensing actions, we can restrict our attention to entailments of $\Sigma_0 \wedge \Sigma_{dyn}$. Note that because Σ_0 might include disjunctions (and possibly quantifiers), there might be multiple worlds where Σ_0 is true. For example, in a propositional language with only two propositions $\{p,q\}$, $\Sigma_0 = (p \wedge \neg q)$ means there is only a single world where Σ_0 is true initially, but $\Sigma_0 = (p \vee q)$ means that there are three worlds where Σ_0 is true initially. In other words, Σ_0 might correspond to a single or multiple initial states in classical planning parlance.

If we are wanting to model knowledge, the agent cannot be expected to know everything that is true, and so let Σ_0' be what is believed initially. It may seem natural to let $\Sigma_0' \subseteq \Sigma_0$, but that it not necessary. The agent might be uncertain about what is true (e.g., Σ_0 might have p but Σ_0' has $p \vee q$ instead).[5] However, for simplicity, we will require that agents at least believe the dynamics works as would the real world. Therefore, we consider entailments wrt the following *background theory*:

$$\Sigma = \Sigma_0 \wedge \Sigma_{dyn} \wedge \boldsymbol{O}(\Sigma_0' \wedge \Sigma_{dyn}). \tag{1}$$

There are conveniences afforded by a basic action theory of this form. Firstly, as far as a non-epistemic account of planning is concerned (that is, one where the knowing modality is not present in the goal), we would be checking the entailment of non-modal goal formulas, and therefore, it is immediate that only $\Sigma_0 \wedge \Sigma_{dyn}$ from Σ is involved. Everything in the context of an epistemic operator in Σ can be ignored. This is precisely what we will explore in the first set of results on counterfactual plans. But when we need to refer to formulas involving epistemic modalities, we will only need to refer to the non-objective parts of Σ. (When the agent performs sensing actions, however, they will provide values from

[5] If the agent believes facts that are conflicted by observations about the real world, beliefs may need to be revised [10], a matter we ignore for now. Our theory of knowledge is based on *knowledge expansion* where sensing ensures that the agent is more certain about the world [35, 34]. In the case of reconciliation-based explanations, however, we will need to entertain a simple type of revision based on the deletion of facts from Σ_0', as we shall shortly see. A general treatment of deleting in first-order languages might be based on *forgetting* [28].

Σ_0.) Thus, we can simply concern ourselves with entailments of Σ henceforth.

Example Let us consider a simple blocks world example, involving picking up and dropping objects, but also quenching (rapidly cooling to very low temperatures) objects so that they become fragile, adapted from [24, 23]. As usual, picking up is only when possible when the robot is not already holding anything, and dropping is only possible when it is already holding the object. Also, broken objects in the robot's hand can be repaired. So,

$$\begin{aligned}\Box Poss(a) \equiv\ & (a = pickup(x) \wedge \forall z.\neg Holding(z)) \vee (a = drop(x) \wedge Holding(x)) \vee \\ & (a = quench(x) \wedge Holding(x)) \vee (a = repair(x) \wedge Holding(x) \wedge Broken(x)).\end{aligned}$$

Let us also permit a sensing axiom that allows one to look up if an object is made of glass:

$$\Box SF(a) \equiv (a = isGlass(x) \wedge Glass(x)) \vee a \neq isGlass(x).$$

To now consider successor state axioms, let us suppose holding an object is possible by picking it up. A fragile object gets broken on dropping it, and not repairing it. Quenching makes an object fragile, regardless of whether it was previously fragile or not. An object being a glass is a rigid property. These are formalized as the axioms below, where the left hand side of the equivalence captures the idea that for every sequence of actions, the effect of doing a on a predicate is given by the right hand side of the equivalence. These capture Reiter's monotonic solution to the frame problem using successor state axioms, but now in $\mathscr{ES}$.

$$\begin{aligned}&\Box[a]Holding(x) \equiv a = pickup(x) \vee (Holding(x) \wedge a \neq drop(x)).\\ &\Box[a]Broken(x) \equiv (a = drop(x) \wedge Fragile(x)) \vee (Broken(x) \wedge a \neq repair(x)).\\ &\Box[a]Fragile(x) \equiv Fragile(x) \vee a = quench(x).\\ &\Box[a]Glass(x) \equiv Glass(x).\end{aligned}$$

Let us suppose the initial theory only-believed by the agent is the following, where nothing is held, there is a non-broken object c made of glass, and as one would assume, glass objects are fragile:

$$\Sigma_0' = \{Glass(c), \neg\exists x Holding(x), \neg Broken(c), \forall x(Glass(x) \supset Fragile(x))\}.$$

In the real world, let us additionally suppose there is another glass object d but also a non-fragile object h:[6] $\Sigma_0 = \Sigma_0' \cup \{(Glass(d)), (\neg Glass(h)), (\neg Fragile(h))\}$.

That is, whatever the agent believes happens to be true in the real world, but the agent does not know about d being made of glass and h not being fragile. Σ_0 in itself does not commit to how many objects there are in the universe, and so it should be clear to the agent that there are (possibly infinitely) many objects outside of c for which it is not known whether they are fragile or made of glass, for example.

Here a few examples of entailments of Σ: (a) $(\neg \boldsymbol{K} Glass(d) \wedge \neg \boldsymbol{K} \neg Glass(d))$; (b) $\boldsymbol{K}\neg\boldsymbol{K} Glass(d)$; (c) $[isGlass(h)]\boldsymbol{K} Glass(d)$; and (d) $[isGlass(h)]\boldsymbol{K}\boldsymbol{K}\boldsymbol{K} Glass(d)$.

That is, the agent's initial beliefs imply that the agent does not know whether d is made of glass. Moreover, by introspection, the agent knows that it does know if d is made of glass. But after sensing h for glass, it knows that it knows that it knows (and so on arbitrarily) that d is made of glass.

[6] We use the set notation and the formula notation (that is, using conjunctions of formulas) for theories as per convenience.

3 Reasoning & Planning

Given a background theory Σ, an action sequencen $\delta = a_1 \cdots a_k$, and a (non-modal) goal formula ϕ, the classical problem of *projection* [34] is to identify if the sequence enables ϕ. That is, whether $\Sigma \models [\delta]\phi$. We also want to ensure that the action sequence is executable (aka *valid* and/or *legal*). So let us $Exec(\langle\rangle) = true$, and $Exec(a \cdot \delta) = Poss(a) \wedge [a]Exec(\delta)$. Then, we check: $\Sigma \models Exec(\delta) \wedge [\delta]\phi$. In the epistemic setting [35], we are interested in checking if ϕ is known after executing δ, which might include sensing actions too. That is, whether[7] $\Sigma \models [\delta]\boldsymbol{K}\phi$. When adding action executability, we would have: $\Sigma \models [\delta]\boldsymbol{K}\phi \wedge \boldsymbol{K}Exec(\delta)$. So the agent also knows that the sequence is executable: which means δ is executable in every world in the agent's epistemic state. Note that because we do not require the real world to be necessarily included in the epistemic state, it is not necessarily that δ is actually executable in the real world. If we needed to additionally enforce that, we would need: $\Sigma \models Exec(\delta) \wedge [\delta]\boldsymbol{K}\phi \wedge \boldsymbol{K}Exec(\delta)$.

The task of planning, then, is to identify a sequence δ such that ϕ is made true (in the non-epistemic setting) or that ϕ is known to be true, and that the appropriate executability condition holds.

Note that, we do not require plans to simply be a sequence of (physical) actions. For one thing, they may involve sensing actions, based on which the agent obtains information about the world. For another, we might be interested plans involving recursion [41], conditional statements and tests [27, 25, 15]. Such plan structures do not change the nature of the reasoning problem, however: no matter the plan structure, we will be evaluating if the sequence of actions executed by the agent enables some goal, that is, whether the structure instantiates a sequence δ such that $\Sigma \models [\delta]\phi$ and $\Sigma \models [\delta]\boldsymbol{K}\phi$ for world-state and epistemic planning respectively. Likewise, regression is not limited to only action sequences and can work with conditional and recursive plans.[8]

Finally, it can be shown that reasoning about actions and knowledge can be reduced to non-modal reasoning. We omit the details but refer readers to [23]. (We will included an extended report with some examples.) With a finite domain assumption, this can be further reduced to propositional reasoning.

4 Counterfactual Explanations

We will firstly attempt to characterize counterfactual (CF) explanations as plans, at an objective level. This could be viewed, therefore, as an instance of planning with incomplete information, but it can also be ultimately linked to the epistemic setting, as we shall see below. Simply put, a CF explanation is an alternative course of action that negates the goal. (Conversely, if some sequence does not enable the goal, we search for an explanation that does.[9])

[7] It might also be of interest to know whether ϕ is true [12], that is, checking that $\Sigma \models [\delta](\boldsymbol{K}\phi \vee \boldsymbol{K}\neg\phi)$. Here, the second disjunct is asserting that the agents knows ϕ to be false.

[8] In the expressive programming formalism of GOLOG [25], for example, we provide the semantics for program execution such that there is a history (an action sequence) that terminates the program in addition to satisfying the goal [15]. Likewise, with loopy plans, we provide the semantics for plan execution such that there is a history that reaches the final state of the plan structure in addition to goal satisfaction [27].

[9] In relation to machine learning [43], the idea is to produce an action sequence that changes the outcome. For example, if ϕ represents an applicant getting rejected for a job application, then we find a plan to ensure that they are accepted. Conversely, if ϕ states that moving an object to a different room causes it to break, we find a plan to ensure that the object is not broken during the move. In another scenario, if ϕ states that high-risk individuals have their loan approved, we might be interested in additional assumptions that ensure that such individuals are mostly denied loans unless further constraints hold.

Thus, it is not the polarity of the formula that is relevant here, and our use of the term "goal" is perhaps slightly misleading. Essentially, our definitions below formalize the identification of conditions and sequences that toggle the outcome.

Definition 1. *Suppose* $\Sigma \models Exec(\delta) \wedge [\delta]\phi$. *A CF explanation for* ϕ *after* δ *is an action sequence* δ' *such that* $\Sigma \models Exec(\delta') \wedge [\delta']\neg\phi$ *and* $dist(\delta', \delta)$ *is minimal.*

One natural candidate for the distance metric is the cost of actions (and hence the cost of plans) [42]. Let us explore some measures below that does not necessitate associating explicit numbers with actions for simplicity. Of course, the appropriate measure might very well depend on the application domain.

Definition 2. *Given two sequences* δ, δ', *define length-based minimality as minimizing* $|(length(\delta') - length(\delta))|$, *which is the absolute value of the difference in lengths. Length is defined inductively:* $length(\langle\rangle) = 0$, *and* $length(\delta \cdot a) = length(\delta) + 1$.

Example 3. *Suppose* $\delta = pickup(c) \cdot drop(c) \cdot repair(c)$ *and the goal is* $\neg Broken(c)$. *The shortest CF explanation for* $Broken(c)$ *is* $\delta' = pickup(c) \cdot drop(c)$. *That is: (a)* $\Sigma \models [\delta]\neg Broken(c)$; *and (b)* $\Sigma \models [\delta']Broken(c)$.

Let us consider another measure based on all the properties of the world that are affected. For any δ, define $fluents(\delta)$ as the set of all fluents mentioned in the successor state and precondition axioms of actions in δ.

Definition 4. *Given* δ, δ' *as above, define fluent-based minimality as minimizing* $|size(fluents(\delta)) - size(fluents(\delta'))|$.

Example 5. *Suppose* $\delta = pickup(h) \cdot drop(h)$ *for goal* $\neg Broken(h)$. *The fluent set for* δ *is* $\{Holding(x), Broken(x)\}$. *Because h is not fragile, we would need to quench it. Consider that the fluent set for* $\delta' = pickup(h) \cdot quench(h) \cdot drop(h)$ *is* $\{Holding(x), Fragile(x), Broken(x)\}$, *and so it is minimally larger: that is, there is no other* δ' *with the same fluent set as* δ *which achieves* $Broken(h)$. *As desired,* $\Sigma \models [\delta']Broken(h)$.

As it turns out, only optimizing for the affected set is not quite right because many irrelevant ground actions could be included.

Definition 6. *Given* δ, δ' *as above, define plan-and-effect minimality as jointly minimizing both length-based and fluent-based minimality.*

Example 7. *It should be clear that only optimizing for fluent-based minimality is problematic. Consider once more,* $\delta = pickup(h) \cdot drop(h)$ *for goal* $\neg Broken(h)$. *Let* $\delta'' = pickup(d) \cdot drop(d) \cdot \delta'$, *where* $\delta' = pickup(h) \cdot quench(h) \cdot drop(h)$. *The fluent set of* δ'' *does not differ from that of* δ *much more than that of* δ' *does. However,* δ'' *has some irrelevant actions for achieving* $Broken(h)$. *Thus,* δ' *achieves plan-and-effect minimality.*

An important additional ingredient with counterfactual explanations is *diversity* [29], where we might seek multiple CF explanations but constrained according some feature. For example, we could be looking for students whose scored low in mathematics (the constraint) while still graduating (the latter being the goal), looking for tall students (the constraint) who still do not play basketball well (the goal), and so on. Properties such as people being tall can be modelled as rigid predicates, but our definition does not limit itself to rigids.

Definition 8. *Given* Σ, δ, ϕ *as above,* $k \in \mathbb{N}$, *and any non-modal formula* α *as a diversity constraint, a diverse CF explanation is a sequence* δ' *such that* $\Sigma \models Exec(\delta') \wedge [\delta'](\alpha \wedge \neg\phi)$ *and* $dist(\delta', \delta) \leq k$.

Example 9. *Suppose* $\delta = pickup(h) \cdot drop(h)$ *for goal* $\phi = \exists x \neg Broken(x)$. *Suppose we are interested in a broken object, but with the diversity constraint of it being made of glass. In other words,* $\alpha = \exists x Glass(x)$, *and so we are to find a sequence* δ' *such that* $\exists x(Glass(x) \wedge Broken(x))$ *is made true. It is easy to see that* $\delta' = pickup(c) \cdot drop(c)$ *is such an explanation.*

This then leads to multiple explanations that are close enough.

Definition 10. *Let k be any positive integer denoting the closeness upper bound. Given* Σ, δ, ϕ *as above, and any non-modal formula* α *as a diversity constraint, diverse CF explanations is a set of sequences* $\{\delta_1, \ldots, \delta_n\}$ *such that* $\Sigma \models Exec(\delta_i) \wedge [\delta_i](\alpha \wedge \neg\phi)$ *for every i and* $dist(\delta_i, \delta) \leq k$.

5 Reconciliation-based Explanations

The simplest case of an agent providing a counterfactual explanation is that we formulate plans in the context of knowledge, and so goals can involve nested beliefs.

Definition 11. *Suppose $\Sigma \models [\delta]\boldsymbol{K}\phi \wedge \boldsymbol{K}Exec(\delta)$, where ϕ might mention $\boldsymbol{K}$ but no other modality. Then a counterfactual explanation is δ' such that $dist(\delta', \delta)$ is minimal and $\Sigma \models [\delta']\boldsymbol{K}\neg\phi \wedge \boldsymbol{K}Exec(\delta')$.*

Recall that we are not seeking $[\delta]\neg\boldsymbol{K}\phi$, because this is the case of an agent being ignorant. We instead seek δ' after which the agent knows that ϕ is false.

Note that in the above definition we were not stipulating executability in the real world, because it suffices for the account to be purely epistemic. We can enforce this additionally, of course, but it will come up naturally for the definitions below because the user needs to make sure she is only suggesting legal actions.

Example 12. *Following our examples above, consider $\delta = pickup(c) \cdot drop(c)$ and clearly Σ entails $[\delta]\boldsymbol{K}Broken(c)$. The explanation $\delta' = \delta \cdot repair(c)$ achieves $\boldsymbol{K}\neg Broken(c)$.*

What we will consider below is the case where the user assists in achieving goals. So this leads to a broader notion of counterfactuals: consider an alternate history or additional knowledge such that the goal becomes true.

5.1 Agents Only-Knowing Partial Truths

For this case of ignorant agents, we assume $\Sigma'_0 \subseteq \Sigma_0$.[10] Suppose a plan δ fails in achieving $\boldsymbol{K}\phi$. A CF explanation here amounts to considering a possible world and a sequence such that the agent knows ϕ. So either there are missing actions, or missing knowledge, or both.

Definition 13. *(Missing actions.) Suppose $\Sigma \models Exec(\delta) \wedge \boldsymbol{K}Exec(\delta)$ but $\Sigma \not\models [\delta]\boldsymbol{K}\phi$, that is, $\Sigma \models [\delta]\neg\boldsymbol{K}\phi$. Suppose there is a sequence δ' such that $dist(\delta', \delta)$ is minimal, $\Sigma \models [\delta']\phi \wedge Exec(\delta')$, and $\Sigma \models [\delta']\boldsymbol{K}\phi \wedge \boldsymbol{K}Exec(\delta')$. Then a CF explanation is δ'.*

Note that we do not insist $\Sigma \models [\delta]\phi$, because as the definition title suggests, there might actions missing. One might also wonder why we insist on the agent also needing to know ϕ after δ': is it not redundant? The answer is no. Firstly, notice that even if there is δ' such that $\Sigma \models [\delta']\phi$, it is not necessary that δ' is minimally away from δ. For example, as far as entailment of objective formulas is concerned, the presence of sensing actions in δ' is irrelevant: sensing only affects the knowledge of the agent and does not affect the real world. But the agent may very well need sensing actions to learn more about the world. Therefore, we insist that we need to find a δ' that is minimally different to δ, enables ϕ in the real world but also enables the knowing of ϕ. In other words, if both δ' and δ'' enable ϕ and they differ only in that δ' includes sensing actions whereas δ'' does not, then we want δ' to be the explanation.

Example 14. *Consider $\delta = pickup(c)$ for the goal $\neg\boldsymbol{K}Broken(c)$. The explanation is $\delta' = \delta \cdot drop(c)$, and indeed, $\Sigma \models [\delta']Broken(c)$ but also $\Sigma \models [\delta']\boldsymbol{K}Broken(c)$.*

Example 15. *Suppose $\delta = pickup(d) \cdot drop(d)$ for the goal $\neg\boldsymbol{K}Broken(d)$. But in fact, $\Sigma \models [\delta]Broken(d)$, and so the agent does not know d is broken owing to the fact that it does not know that d is made of glass. So consider $\delta' = pickup(d) \cdot isGlass(d) \cdot drop(d)$. This does not affect what is true in the world, but does lead the agent to know that $Glass(d)$. Therefore, δ' is the explanation since $\Sigma \models [\delta']Broken(d)$, and $\Sigma \models [\delta']\boldsymbol{K}Broken(d)$.*

[10] As mentioned before, we do not require $\Box(\boldsymbol{K}\alpha \supset \alpha)$ to be valid, but if this was stipulated in the logic (by insisting that the real world $w \in e$), then it should always be that $\Sigma_0 \models \Sigma'_0$. (If not, then $\boldsymbol{K}\Sigma'_0 \supset \Sigma'_0$ would be falsified in the real world.)

Definition 16. *(Missing knowledge.) Suppose* $\Sigma \models Exec(\delta)$*. Suppose* $\Sigma \not\models [\delta]\boldsymbol{K}\phi \wedge \boldsymbol{K}Exec(\delta)$ *but* $\Sigma \models [\delta]\phi$*. Then suppose there is some* $\alpha \in \Sigma_0 - \Sigma'_0$ *such that* $\Sigma_0 \wedge \Sigma_{dyn} \wedge \boldsymbol{O}(\Sigma'_0 \wedge \alpha \wedge \Sigma_{dyn}) \models [\delta]\boldsymbol{K}\phi \wedge \boldsymbol{K}Exec(\delta)$*. Then the smallest such* α *is the explanation.*

Note that we do not assume in the definition that $\Sigma \models \boldsymbol{K}Exec(\delta)$, because such an α could be necessary knowledge to reason about the executability of actions.

Example 17. *Given* $\delta = pickup(d) \cdot drop(d)$*, we know that* $[\delta]\neg\boldsymbol{K}Broken(d)$*. But for* $\alpha = Glass(d)$*, and owing to the fact that every object made of glass is declared to be fragile in* Σ'_0*, we see that* $\Sigma_0 \wedge \Sigma_{dyn} \wedge \boldsymbol{O}(\Sigma'_0 \wedge \alpha \wedge \Sigma_{dyn})$ *entails* $[\delta]\boldsymbol{K}Broken(d)$*. So* α *is the explanation.*

Definition 18. *(Missing knowledge and action.) Suppose* $\Sigma \models Exec(\delta)$ *but* $\Sigma \not\models [\delta]\phi$*, or* $\Sigma \not\models [\delta]\boldsymbol{K}\phi$*. Suppose there is a minimally distant* δ' *and some* α *such that* $\Sigma_0 \wedge \Sigma_{dyn} \wedge \boldsymbol{O}(\Sigma'_0 \wedge \alpha \wedge \Sigma_{dyn})$ *entails:* $[\delta'](\phi \wedge \boldsymbol{K}\phi) \wedge Exec(\delta') \wedge \boldsymbol{K}Exec(\delta')$*. Then the smallest such* α *together with* δ' *is the explanation.*

Note that, firstly, if $\Box(\boldsymbol{K}\alpha \supset \alpha)$ was true in the logic, $\Sigma \not\models [\delta]\phi$ also means $\Sigma \not\models [\delta]\boldsymbol{K}\phi$, because the real world $w \in e$. By assumption, if ϕ is not made true after δ, then the agent cannot come to know ϕ after δ. Since we do not require knowledge being true, the definition has to make stipulations about both $[\delta]\phi$ and $[\delta]\boldsymbol{K}\phi$. Moreover, it is possible that $\Sigma \models [\delta]\phi$ but $\Sigma \not\models [\delta]\boldsymbol{K}\phi$ because there are sensing actions that could enable the agent to learn sufficient information for knowing ϕ after δ, or because there is some information that cannot be accessed by sensing that needs to be added to Σ'_0 for the agent to infer $\boldsymbol{K}\phi$ after δ, (or both).[11]

Example 19. *Let us assume quenching comes with an additional condition that it only works with metals:* $\Box Poss(quench(x)) \equiv Holding(x) \wedge Metal(x)$*. Let* Σ'_0 *be as before, and let the initial state of the world be given by* $\Sigma^*_0 = \Sigma'_0 \cup \{\neg Fragile(h), \neg Glass(h), Metal(h)\}$*. Consider* $\delta = pickup(h) \cdot drop(h)$*, and it is easy to see that* $\Sigma^*_0 \wedge \Sigma_{dyn} \wedge \boldsymbol{O}(\Sigma'_0 \wedge \Sigma_{dyn})$ *entails* $[\delta]\neg\boldsymbol{K}Broken(h)$*. Consider* $\delta' = pickup(h) \cdot quench(h) \cdot drop(h)$ *which adds the quenching action. In itself, the sequence is not known to be executable because the agent does not know that h is metallic. So* $\alpha = Metal(h)$ *together with* δ' *is the explanation because* $\Sigma^*_0 \wedge \Sigma_{dyn} \wedge \boldsymbol{O}(\Sigma'_0 \wedge \alpha \wedge \Sigma_{dyn})$ *entails* $[\delta'](\phi \wedge \boldsymbol{K}\phi)$ *where* $\phi = Broken(h)$*, as well as the legality of the sequence and knowledge of its legality.*

5.2 Agents Only-Knowing Weakened Truths

Here we assume $\Sigma'_0 \not\subseteq \Sigma_0$ but for every $\alpha \in \Sigma'_0$, $\Sigma_0 \models \alpha$. In other words, $\Sigma_0 \models \Sigma'_0$. That is, we might have an atom $p \in \Sigma_0$, but Σ'_0 instead has $(p \vee q)$. We then can define an account involving missing knowledge and actions, and so the same definition from 18 applies.

Example 20. *Let us consider Example 19 except that the initial theory of the agent is* $\Sigma''_0 = \Sigma'_0 \wedge (Metal(h) \vee Metal(d))$*. That is, it includes all the formulas from* Σ'_0 *but also information that h is metallic or (falsely) that d is metallic. However,* $\Sigma^*_0 \models (Metal(h) \vee Metal(d))$*, and so* α *and* δ' *from Example 19 counts as the explanation.*

---

[11] This suggests a "criteria" for triggering the addition of knowledge. Define two sequences δ and δ' to be close iff δ' only differs from δ in having sensing actions. Condition knowlege addition only when there is no δ' that is close in this sense, and $\Sigma \models [\delta]\phi \wedge [\delta]\neg\boldsymbol{K}\phi \wedge [\delta']\boldsymbol{K}\phi$ along with $\Sigma \models Exec(\delta) \wedge Exec(\delta') \wedge \boldsymbol{K}Exec(\delta) \wedge \boldsymbol{K}Exec(\delta')$. So, if there is a legal sequence that only augments δ with sensing but enables knowing the goal, then we conclude that no new knowledge needs to be added. If there are multiple such augmented sequences δ' and δ'', we would choose the shortest such sequence. This would avoid applying sensing actions arbitrarily or using sensors that do not inform the agent about anything relevant for ϕ.

5.3 Agents with False Beliefs

False beliefs are only satisfiable when $\square(\boldsymbol{K}\alpha \supset \alpha)$ is not valid, as it happens to be in our case. For simplicity, we deal with the case of missing knowledge, and this can be easily coupled with missing actions in the manner discussed above. (Our example will deal with both.)

Definition 21. *Suppose $\Sigma_0' \not\subseteq \Sigma_0$, and moreover $\Sigma_0 \not\models \Sigma_0'$. Suppose $\Sigma \not\models ([\delta]\boldsymbol{K}\phi \wedge \boldsymbol{K}Exec(\delta))$ but $\Sigma \models Exec(\delta) \wedge [\delta]\phi$. Suppose there is $\alpha \in \Sigma_0$ and $\beta \in \Sigma_0'$ such that $\Sigma_0 \cup \Sigma_{dyn} \cup \boldsymbol{O}((\Sigma_0' - \{\beta\}) \cup \{\alpha\} \cup \Sigma_{dyn}) \models [\delta]\boldsymbol{K}\phi \wedge \boldsymbol{K}Exec(\delta)$. Then the smallest such α and β are the explanations.*

Example 22. *Consider Example 19 and let $\Sigma_0'' = \Sigma_0' \wedge \neg Metal(h)$. So the agent only-knows everything from Σ_0' as well as a false fact about h. Let Σ_0^*, δ, and δ' be as in Example 19. Now note that given $\alpha = Metal(h)$, $\beta = \neg Metal(h)$ and δ', we have $\Sigma_0^* \wedge \Sigma_{dyn} \wedge \boldsymbol{O}(\Sigma_0' \wedge \alpha \wedge \Sigma_{dyn})$ entails $[\delta'](\phi \wedge \boldsymbol{K}\phi) \wedge Exec(\delta') \wedge \boldsymbol{K}Exec(\delta')$, for $\phi = Broken(h)$. As it turns out Σ_0' is obtained by removing β from Σ_0'' by construction, and so α, β and δ' constitutes as the explanation.*

5.4 Possibility vs Knowledge

In many applications, we may not require that the agent knows ϕ, only that it considers ϕ possible. In the explanation generation framework of [42], for example, there is a notion of credulous entailment where a single belief state suffices to checking the validity and achievement of a plan. (In contrast, skeptical entailment is when every belief state is involved.) The analogous notion in an epistemic language is to introduce a companion modal operator $\boldsymbol{B}$ with the following semantics:

- $e, w, z \models \boldsymbol{B}\alpha$ iff there is some $w' \sim_z w, w' \in e$ such that $e, w', z \models \alpha$.

As long as there is at least one world where α is true, $\boldsymbol{B}\alpha$ is evaluated to true at (e, w, z). This is a companion modal operator to $\boldsymbol{K}$ for *possibility.* We might introduce an analogue to Definition 18 in using $\boldsymbol{B}$ instead of $\boldsymbol{K}$ as the modality in the goal. To see how this works, let us revisit Example 19.

Example 23. *Consider the modified precondition axiom for quench(x), and let Σ_0', Σ_0^*, δ and δ' be as in Example 19. By only-knowing Σ_0', the agent considers some worlds where h is metallic, and others where it is not. Thus, we now see that we do not really need to suggest $\alpha = Metal(h)$ to the agent if the weaker notion of a CF explanation is considered. Indeed, $\Sigma_0^* \wedge \Sigma_{dyn} \wedge \boldsymbol{O}(\Sigma_0' \wedge \Sigma_{dyn})$ entails $[\delta'](\phi \wedge \boldsymbol{B}\phi)$ for $\phi = Broken(h)$.*

In other words, we have augmented actions but we did not need to augment knowledge in the above example. Had the agent believed false things, then we might have needed to augment both, and so can appeal to Definition 21 but using $\boldsymbol{B}$ instead of $\boldsymbol{K}$ to allow for credulous-type reasoning.

6 Other related efforts

In addition to the works discussed in previous sections, the following efforts are related.

There is some syntactic (and perhaps intuitive) connection to explanation-based diagnosis. For example, in [38], the idea is to encode the behavior of the system to be diagnosed as a situation calculus action theory, encode observations as situation calculus formulae, and conjecture a sequence of actions to explain what went wrong with the system. (However, they often need to model faulty or abnormal components when defining the notion of a diagnosis.) In our setting, in contrast, we identify actions and/or knowledge that determine how an outcome can be changed. Nonetheless, we believe that a further formal study to relate such accounts would be useful, and could nicely complement empirical works such as [9]. See also Ginsberg [17].

We previously discussed the reduction of projection and reasoning about knowledge to non-modal reasoning [23], but we did not elaborate on generating plans. For more information on synthesizing plans, programs, and epistemic plans, see [8, 11, 33, 2] and their references.

An alternative approach to computing properties and plans is through answer set programming (ASP) [1, 16], which also supports reasoning about knowledge [13]. In general, our formalization does not preclude consideration of other logical languages. For example, in the simplest setting in this paper, a counterfactual explanation is the synthesis of a course of action that negates the goal or knowledge about the goal. In fact, [5] consider counterfactual explanations for multi-agent systems, that is also motivated in terms of offering an alternative course of action. Although they do not explore a range of definitions with references to knowledge as we do, exploring whether our definitions can be implemented in such approaches is worthwhile.

Such a course for formalisation may help better relate our efforts to declarative approaches to counterfactual explanations. For example, [4] explores the use of ASP for generating counterfactual explanations, but in a classical machine-learning sense, determined by how much certain features affect the overall prediction (understood as a causal link).

7 Conclusions

We developed an account of counterfactual explanations and reconciliation-based counterfactual explanations in this paper. This allows for a simple and clear specification in the presence of missing actions, partial knowledge, weakened beliefs and false beliefs. Existing accounts of discrepancy in plans, among others, can be seen as variations of this more general specification. For the future, it would be interesting to incorporate other notions in our formalization, such as operational aspects of plans, costs, optimality and conciseness [14, 40, 42], towards a unified mathematical specification of explainable planning.

References

[1] C. Baral & M. Gelfond (2005): *Logic programming and reasoning about actions*. In: Handbook of Temporal Reasoning in Artificial Intelligence, Elsevier, pp. 389–426, doi:10.1016/S1574-6526(05)80015-X.

[2] Chitta Baral, Thomas Bolander, Hans van Ditmarsch & Sheila McIlrath (2017): *Epistemic Planning (Dagstuhl Seminar 17231)*. Dagstuhl Reports 7(6), pp. 1–47, doi:10.4230/DagRep.7.6.1. Available at `http://drops.dagstuhl.de/opus/volltexte/2017/8285`.

[3] V. Belle & G. Lakemeyer (2014): *Multiagent Only Knowing in Dynamic Systems*. Journal of Artificial Intelligence Research 49, doi:10.1613/jair.4192.

[4] Leopoldo Bertossi (2021): *Declarative approaches to counterfactual explanations for classification*. Theory and Practice of Logic Programming, pp. 1–35, doi:10.1017/s1471068421000582.

[5] 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, doi:10.1017/S1471068420000320.

[6] Rita Borgo, Michael Cashmore & Daniele Magazzeni (2018): *Towards providing explanations for AI planner decisions*. arXiv preprint arXiv:1810.06338.

[7] Michael Cashmore, Anna Collins, Benjamin Krarup, Senka Krivic, Daniele Magazzeni & David Smith (2019): *Towards explainable AI planning as a service*. arXiv preprint arXiv:1908.05059.

[8] Jens Classen, Patrick Eyerich, Gerhard Lakemeyer & Bernhard Nebel (2007): *Towards an Integration of Golog and Planning*. In: IJCAI, pp. 1846–1851.

[9] Xinyue Dai, Mark T Keane, Laurence Shalloo, Elodie Ruelle & Ruth MJ Byrne (2022): *Counterfactual explanations for prediction and diagnosis in xai*. In: *Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 215–226, doi:10.1145/3514094.3534144.

[10] James P. Delgrande & Hector J. Levesque (2012): *Belief Revision with Sensing and Fallible Actions*. In: *Proc. KR*.

[11] H. P. van Ditmarsch, A. Herzig & T. De Lima (2011): *From Situation Calculus to Dynamic Epistemic Logic*. *J. Log. Comput.* 21(2), pp. 179–204, doi:10.1093/logcom/exq024.

[12] Jie Fan, Yanjing Wang & Hans Van Ditmarsch (2015): *Contingency and knowing whether*. *The Review of Symbolic Logic* 8(1), pp. 75–107, doi:10.1017/S1755020314000343.

[13] Jorge Fandinno, Wolfgang Faber & Michael Gelfond (2022): *Thirty years of epistemic specifications*. *Theory and Practice of Logic Programming* 22(6), pp. 1043–1083, doi:10.1017/S147106842100048X.

[14] Maria Fox, Derek Long & Daniele Magazzeni (2017): *Explainable planning*. *arXiv preprint arXiv:1709.10256*.

[15] Christian Fritz, Jorge A. Baier & Sheila A. McIlraith (2008): *ConGolog, Sin Trans: Compiling ConGolog into Basic Action Theories for Planning and Beyond*. In Brewka & Lang, editors: *Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia*, pp. 600–610. Available at `http://www.aaai.org/Library/KR/2008/kr08-059.php`.

[16] M. Gelfond & V. Lifschitz (1993): *Representing action and change by logic programs*. *The Journal of Logic Programming* 17(2-4), pp. 301–321, doi:10.1016/0743-1066(93)90035-F.

[17] Matthew L Ginsberg (1986): *Counterfactuals*. *Artificial intelligence* 30(1), pp. 35–79, doi:10.1016/0004-3702(86)90067-6.

[18] Joseph Y. Halpern & Yoram Moses (1990): *Knowledge and Common Knowledge in a Distributed Environment*. *J. ACM* 37(3), pp. 549–587, doi:10.1145/79147.79161.

[19] Subbarao Kambhampati (2020): *Challenges of Human-Aware AI Systems.* *AI Magazine* 41(3), doi:10.1609/aimag.v41i3.5257.

[20] R. F. Kelly & A. R. Pearce (2008): *Complex Epistemic Modalities in the Situation Calculus*.

[21] Benjamin Krarup, Michael Cashmore, Daniele Magazzeni & Tim Miller (2019): *Model-based contrastive explanations for explainable planning*. *ICAPS 2019 Workshop on Explainable AI Planning (XAIP)*.

[22] G. Lakemeyer & H. J. Levesque (2011): *A semantic characterization of a useful fragment of the situation calculus with knowledge*. *Artificial Intelligence* 175, pp. 142–164, doi:10.1016/j.artint.2010.04.005.

[23] G. Lakemeyer & Hector J. Levesque (2004): *Situations, Si! Situation terms, No!* In: *Proc. KR*, pp. 516–526.

[24] Y. Lespérance, H. J. Levesque, F. Lin & R. B. Scherl (2000): *Ability and Knowing How in the Situation Calculus*. *Studia Logica* 66(1), pp. 165–186, doi:10.1023/A:1026761331498.

[25] H. Levesque, R. Reiter, Y. Lespérance, F. Lin & R. Scherl (1997): *GOLOG: A Logic Programming Language for Dynamic Domains*. *Journal of Logic Programming* 31, pp. 59–84, doi:10.1016/S0743-1066(96)00121-5.

[26] H. J. Levesque (1990): *All I know: a study in autoepistemic logic*. *Artificial Intelligence* 42(2-3), pp. 263–309, doi:10.1016/0004-3702(90)90056-6.

[27] H. J. Levesque (1996): *What Is Planning in the Presence of Sensing?* In: *Proc. AAAI / IAAI*, pp. 1139–1146.

[28] F. Lin & R. Reiter (1994): *Forget it*. In: *Working Notes of AAAI Fall Symposium on Relevance*, pp. 154–159.

[29] Ramaravind K Mothilal, Amit Sharma & Chenhao Tan (2020): *Explaining machine learning classifiers through diverse counterfactual explanations*. In: *Proceedings of the 2020 conference on fairness, accountability, and transparency*, pp. 607–617, doi:10.1145/3351095.3372850.

[30] C. Muise, V. Belle, P. Felli, S. McIlraith, T. Miller, A. Pearce & L. Sonenberg (2015): *Planning Over Multi-Agent Epistemic States: A Classical Planning Approach*. In: *Proc. AAAI*, doi:10.1609/aaai.v29i1.9665.

[31] Judea Pearl (2009): *Causality*. Cambridge university press, doi:10.1017/CBO9780511803161.

[32] E.P.D. Pednault (1989): *ADL: Exploring the middle ground between STRIPS and the situation calculus*. In: *Proc. KR*, pp. 324–332, doi:10.5555/112922.112954.

[33] Daniel Reifsteck, Thorsten Engesser, Robert Mattmüller & Bernhard Nebel (2019): *Epistemic multi-agent planning using monte-carlo tree search*. In: *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, Springer, pp. 277–289, doi:10.1007/978-3-030-30179-8_24.

[34] R. Reiter (2001): *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT Press, doi:10.7551/mitpress/4074.001.0001.

[35] R. B. Scherl & H. J. Levesque (2003): *Knowledge, action, and the frame problem*. *Artificial Intelligence* 144(1-2), pp. 1–39, doi:10.1016/S0004-3702(02)00365-X.

[36] S. Shapiro, Y. Lespérance & H.J. Levesque (2002): *The cognitive agents specification language and verification environment for multiagent systems*. In: *Proc. AAMAS*, pp. 19–26, doi:10.1145/544741.544746.

[37] Maayan Shvo, Toryn Q Klassen & Sheila A McIlraith (2022): *Resolving Misconceptions about the Plans of Agents via Theory of Mind*. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, 32, pp. 719–729.

[38] Shirin Sohrabi, Jorge A Baier & Sheila A McIlraith (2010): *Diagnosis as planning revisited*. In: *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*.

[39] T.C. Son & C. Baral (2001): *Formalizing sensing actions–A transition function based approach*. *Artificial Intelligence* 125(1-2), pp. 19–91, doi:10.1016/S0004-3702(00)00080-1.

[40] Sarath Sreedharan, Tathagata Chakraborti & Subbarao Kambhampati (2018): *Handling model uncertainty and multiplicity in explanations via model reconciliation*. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, 28, pp. 518–526.

[41] Siddharth Srivastava (2010): *Foundations and Applications of Generalized Planning*. Ph.D. thesis, Department of Computer Science, University of Massachusetts Amherst, doi:10.3233/AIC-2011-0508.

[42] Stylianos Loukas Vasileiou, William Yeoh, Tran Cao Son, Ashwin Kumar, Michael Cashmore & Dianele Magazzeni (2022): *A Logic-Based Explanation Generation Framework for Classical and Hybrid Planning Problems*. *Journal of Artificial Intelligence Research* 73, pp. 1473–1534, doi:10.1613/jair.1.13431.

[43] Sandra Wachter, Brent Mittelstadt & Chris Russell (2017): *Counterfactual explanations without opening the black box: Automated decisions and the GDPR*. *Harv. JL & Tech.* 31, p. 841.

Abduction of Domain Relationships from Data for VQA

Al Mehdi Saadat Chowdhury, Paulo Shakarian
School of Computing and Augmented Intelligence
Arizona State University
Tempe, Arizona, USA
{achowd43,pshak02}@asu.edu

Gerardo I. Simari
DCIC Univ. Nac. del Sur (UNS)
ICIC (UNS-CONICET)
Bahia Blanca, Argentina
gis@cs.uns.edu.ar

In this paper, we study the problem of visual question answering (VQA) where the image and query are represented by ASP programs that lack domain data. We provide an approach that is orthogonal and complementary to existing knowledge augmentation techniques where we abduce domain relationships of image constructs from past examples. After framing the abduction problem, we provide a baseline approach, and an implementation that significantly improves the accuracy of query answering yet requires few examples.

1 Introduction

Visual Question Answering (VQA) is an AI task designed to reason about images. Commonly, the image is transformed into a "scene graph" that enables the deployment of more formal reasoning tools. For example, in recent work, both the scene graph and associated query were represented as an ASP Program [2, 1]; however, notably the scene graph itself only contains information about the scene, but lacks commonsense knowledge – in particular, knowledge about the domains of attributes identified by the scene. Existing work to address this shortcoming relies on leveraging large commonsense knowledge graphs for obtaining domain knowledge [5, 6, 7]. However, such approaches require the ability to accurately align the language of the knowledge graph with the language of the scene graph. Further, for some applications, this does not guarantee that the aligned knowledge graph will necessarily improve VQA performance (e.g., if domain knowledge relevant to the queries is not possessed in the knowledge graph). In this paper, we provide an orthogonal and complementary approach that leverages logical representations of the scene graph and query to abduce domain relationships that can improve query answering performance. We frame the abduction problem and provide a simple algorithm that provides a valid solution. We also provide an implementation and show on a standard dataset that we can improve question answering accuracy from 59.98% to 81.01%, and provide comparable results with few historical examples.

Motivating Example. Consider the simple scene graph depicted in Figure 1 and the query "*What is the color of the fruit to the right of the juice?*". Without the shaded nodes (which indicate domain information external to the image) there is no attribute of any constant associated with *banana* that is associated with the domain *color* or the domain *fruit*. Hence, the only answer would be to assume that there is no fruit or the color information is not given, or randomly guess *large* (while not a color, it is an attribute) or *yellow*. In this paper, we will look to abduce these domain relationships from a limited number of examples.

2 Technical Preliminaries

We extend the framework of [2], which represents both images and queries as ASP programs (and the programs can be directly represented as an equivalent scene graph as shown in Figure 1). Their approach

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 168–174, doi:10.4204/EPTCS.416.15

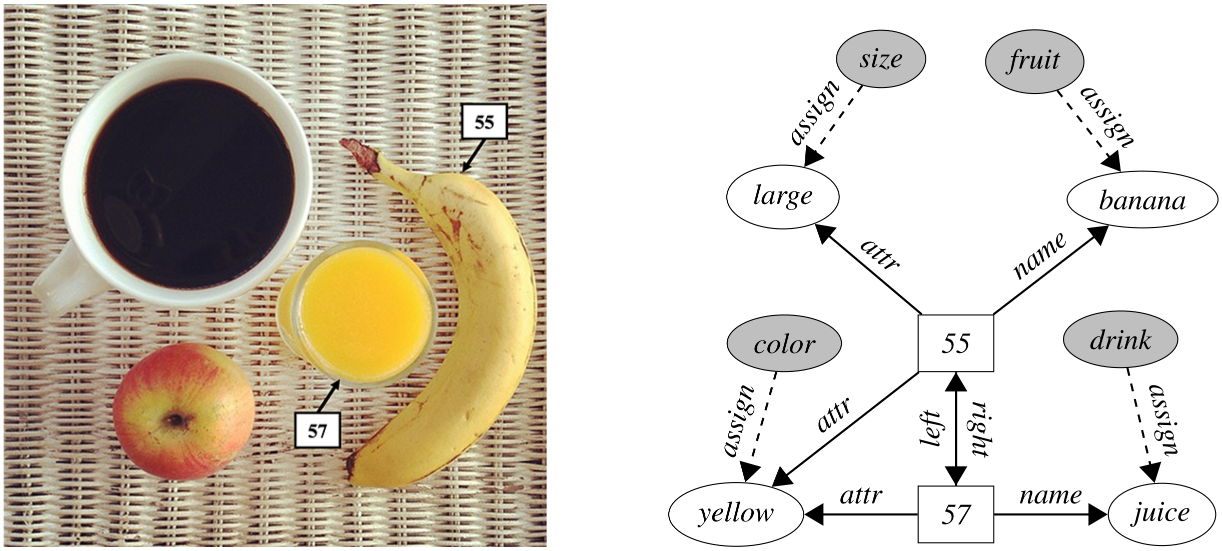


Figure 1: An image (left) and a section of its corresponding scene graph (right). In the scene graph, square nodes represent objects, oval nodes represent attributes, and solid edges connect objects to attributes. Shaded nodes represent domain knowledge, connected to attributes by dashed edges.

to VQA leverages a neurosymbolic framework and was tested on synthetic datasets (e.g., CLEVR [4]) that involve limited objects and attributes. We seek to extend their results to real-world datasets such as GQA [3], which are more complex. We follow the logic programming construct as [2] in that we have logical facts representing the scene graphs (Π^I), the query to be answered (Π^Q), as well as standard "VQA helper" rules (Π^R).

We assume the existence of a first order logical language (constants $\mathscr{C}$, variables $\mathscr{V}$, predicates $\mathscr{P}$). Set $\mathscr{C}$ has several subsets: objects ($\mathscr{C}_{obj}$), attributes ($\mathscr{C}_{att}$), domains ($\mathscr{C}_{dom}$), and single choice questions ($\mathscr{C}_{sinChoice}$). Additionally, we will have a special binary predicate *assign* where the first argument is an attribute and the second is a domain. Every attribute can thus be associated with one or more domains via atom $assign(a,d)$, meaning that attribute a has domain d. We will also define Answer Set Programming (ASP) rules in the usual manner; a rule with no body is a fact and a set of rules is a program. Given a program Π, the subset of facts in Π where the head is formed with *assign* is called the "domain relationships", and denoted Π^D. Likewise, we assume programs representing an image and a query, Π^I and Π^Q, respectively, that do not contain domain relationships, and a common set of rules Π^R that answers the query using Π^I and Π^Q. Also, we shall use the standard ASP semantics based on interpretations [2], and use the notation $I \models \Pi$ to denote that interpretation I satisfies program Π. Further, we say that program $\Pi_1 \models \Pi_2$ (read "Π_1 entails Π_2") meaning that all interpretations that satisfy Π_1 also satisfy Π_2.

In this work, we are primarily concerned with the case where there is a common Π^D for a collection of image-query program pairs ("examples") denoted $\langle \Pi_1^I, \Pi_1^Q \rangle, \ldots, \langle \Pi_n^I, \Pi_n^Q \rangle$. We may also know that a given $\langle \Pi_i^I, \Pi_i^Q \rangle$ is associated with some set of *ground truth* Π_i^{GT}. Due to the lack of domain knowledge, $\Pi_i^I \cup \Pi_i^Q \cup \Pi^R$ may not entail Π_i^{GT}. However if an oracle provides a correct Π^D, we have that $\Pi_i^I \cup \Pi_i^Q \cup \Pi^R \cup \Pi^D \models \Pi_i^{GT}$. We show an example of this case below taken from the scene graph dataset of [3] (depicted in Figure 1), which we also use in our experiments.

Example 2.1. *Consider a program $\Pi_i = \Pi_i^I \cup \Pi_i^Q \cup \Pi^R$ that consists of the following scene representa-*

tion Π_i^I, question representation Π_i^Q for the question "What is the color of the fruit to the right of the juice?", and the set of rules Π^R common to all image-query program pairs:

$$\Pi_i^I = \left\{ \begin{array}{lllll} ob(2317538,51). & name(51,cup). & attr(51,glass). & attr(51,white). & \\ ob(2317538,54). & name(54,apple). & attr(54,round). & attr(54,red). & \\ ob(2317538,55). & name(55,banana). & attr(55,yellow). & attr(55,large). & rel(55,57,right). \\ ob(2317538,57). & name(57,juice). & attr(57,yellow). & rel(57,55,left). & \end{array} \right.$$

$$\Pi_i^Q = \left\{ \begin{array}{lll} scene(0,2317538). & select(1,juice,0). & relate(2,fruit,right,1). \\ query(4,color,3). & exit(5). & \end{array} \right.$$

As in [2], our question representation Π_i^Q is structured so that each query part is organized sequentially, with the first argument of each predicate indicating order and the last argument showing dependency on prior results. This step-by-step approach along with Π^R aids in answering questions effectively:

$$\Pi^R = \left\{ \begin{array}{rcl} r(T,OID) & :- & scene(T,S), ob(S,OID). \\ r(T,OID) & :- & select(T,ON,D), r(D,OID), name(OID,ON). \\ r(T,TID) & :- & relate(T,GC,R,D), r(D,OID), rel(TID,OID,R), name(TID,ON), \\ & & assign(ON,GC). \\ r(T,A) & :- & query(T,color,D), r(D,OID), attr(OID,A), assign(A,color). \\ result(RSLT) & :- & exit(T), r(T-1,RSLT). \\ empty(AT) & :- & exit(T), not\ r(AT,\_), AT = 0..T-1. \end{array} \right.$$

For this question, the ground truth is the program:

$$\Pi_i^{GT} = \{\ result(yellow).\ \}$$

However, due to the lack of atoms $assign(banana, fruit)$ and $assign(yellow, color)$, we see that, $\Pi_i \not\models \Pi_i^{GT}$. Now we assume that an oracle provides us with Π^D, as follows:

$$\Pi^D = \left\{ \begin{array}{lll} assign(glass,material). & assign(white,color). & assign(apple,fruit). \\ assign(round,shape). & assign(red,color). & assign(banana,fruit). \\ assign(yellow,color). & assign(large,size). & assign(juice,drink). \end{array} \right.$$

With the existence of this domain Π^D, now we have $\Pi_i \cup \Pi^D \models \Pi_i^{GT}$.

Fallback Rules. In this framework, where we may have an absent or partial Π^D, it is useful to have "fallback rules" of the form: $assign(att, default) \leftarrow \bigwedge_{att \in \mathscr{C}_{att} \setminus \{default\}} \neg assign(att, DOM)$. This assumes a special attribute constant "default" to which an object without an attribute falls back. The next example augments Example 2.1 with fallback rules:

Example 2.2. *We assume additional fallback rules, added to Π^R, of the form:*

$$\begin{array}{rcl} r(T,A) & :- & query(T,color,D), r(D,OID), attr(OID,A), \\ & & \neg assign(A,color), assign(A,default). \end{array}$$

Returning to our running example, assuming there is no $\{assign(yellow, color).\} \in \Pi^D$, adding fallback rules, we get the following Π^D:

$$\Pi^D = \left\{ \begin{array}{lll} assign(glass,material). & assign(white,color). & assign(apple,fruit). \\ assign(round,shape). & assign(red,color). & assign(banana,fruit). \\ assign(yellow,default). & assign(large,size). & assign(juice,drink). \end{array} \right.$$

Algorithm 1: FAST-DAP

Input : A set of programs $\mathbf{EX} = \{\langle \Pi_1^I, \Pi_1^Q \rangle, \ldots, \langle \Pi_n^I, \Pi_n^Q \rangle\}$ where Π_i^I, and Π_i^Q correspond to scene and question representation; Common Rule Set Π^R with Fallback rules; Set of ground truths $\mathbf{GT} = \{\Pi_1^{GT}, \ldots, \Pi_n^{GT}\}$.

Output: A hypothesis Π^D

$\Pi^D \leftarrow \emptyset$

foreach $\langle \Pi_i^I, \Pi_i^Q \rangle \in \mathbf{EX}$ **do**

  if $choose(w,x,y) \in \Pi_i^Q$ **then** `//` w `is the query type,` x,y `are possible answers`

    if $w \in \mathscr{C}_{sinChoice}$ **then**

      if $result(x) \in \Pi_i^{GT}$ **then** $\Pi^D \leftarrow \Pi^D \cup \{assign(x,w).\}$

      else $\Pi^D \leftarrow \Pi^D \cup \{assign(y,w).\}$

    else

      $\Pi^D \leftarrow \Pi^D \cup \{assign(x,w).\ assign(y,w).\}$

  if $\Pi_i^I \cup \Pi_i^Q \cup \Pi^R \not\models \Pi_i^{GT}$ **then**

    Pick the fact $select(i,c,j) \in \Pi_i^Q$ such that $\Pi_i^I \cup \Pi_i^Q \cup \Pi^R \models empty(i)$ and i is minimal

    if *there does not exist* $name(\_,c) \in \Pi_i^I$ **then** `//` c `is then a general concept`

      Pick $c' \neq c$ such that $name(\_,c') \in \Pi_i^I$ and $\Pi_i^I \cup \Pi_i^Q \cup \Pi^R \cup \{assign(c',c).\} \models \Pi_i^{GT}$

      $\Pi^D \leftarrow \Pi^D \cup \{assign(c',c).\}$

      $support_{c',c}$ += 1

return $\{assign(c',c). \in \Pi^D$ with $support_{c',c} > threshold\}$

Abducing Domain Relationships. We now formalize our problem. Given examples $\mathbf{EX} = \{\langle \Pi_1^I, \Pi_1^Q \rangle, \ldots, \langle \Pi_n^I, \Pi_n^Q \rangle\}$ with a common rule set Π^R (which may or may not include fallback rules) and corresponding ground truth $\mathbf{GT} = \{\Pi_1^{GT}, \ldots, \Pi_n^{GT}\}$, then $\langle \mathbf{EX}, \mathbf{GT}, \Pi^R \rangle$ is a *domain abduction problem* (DAP).

Any Π^D containing only facts formed with *assign* in the head is a *hypothesis* for a DAP. A hypothesis Π^D is an *explanation* for DAP $\langle \mathbf{EX}, \mathbf{GT}, \Pi^R \rangle$ if and only if for all i we have $\Pi_i^I \cup \Pi_i^Q \cup \Pi^R \cup \Pi^D \models \Pi_i^{GT}$. However, when $\mathbf{EX}, \mathbf{GT}$ are noisy (e.g., produced from a machine learning system) there may be no explanation; in such cases, we may be able to find a hypothesis Π^D that maximizes some accuracy or recall metric. For example, finding Π^D that maximizes $\frac{1}{|\mathbf{GT}|}|\{\Pi_i^{GT} \in \mathbf{GT}\ s.t.\ \Pi_i^I \cup \Pi_i^Q \cup \Pi^R \cup \Pi^D \models \Pi_i^{GT}\}|$ (where $|\cdot|$ is set cardinality) would lead to maximized accuracy.

3 A Practical Heuristic Algorithm

In this section, we present a practical, heuristic algorithm for finding a DAP, that while is not guaranteed to maximize the accuracy of question answering, we show to perform very well in practice. There are several reasons as to why we adopt this more practical approach. First, in the general case, a brute-force approach is intractable. Second, even if it is possible to exactly optimize an accuracy metric as described in the previous section, it may still perform poorly when confronted with unseen data due to overfitting. Third, in some cases, the query itself can reveal portions of the ground truth. To address all of these issues, we introduce our practical heuristic algorithm FAST DAP (Algorithm 1). Regarding the first point, the algorithm is highly performant, requiring only one pass over all examples in **EX** – this also allows for trivial parallelization. Second, we only add facts to Π^D that support a certain number of examples, which acts as a form of regularization; we then tune this threshold to maximize accuracy. To address the third point, in lines 3–8 we utilize examples that provide domain information in the query itself (with two answers as in $choose(color, red, blue, 0)$ and with single answer as in $choose(healthy, apple, cake, 0)$), while we leverage the step-by-step nature of the ASP formulation of queries (following [2], see Example 2.1) to identify domain assignments that can satisfy the ground truth (lines 9-14).

4 Evaluation

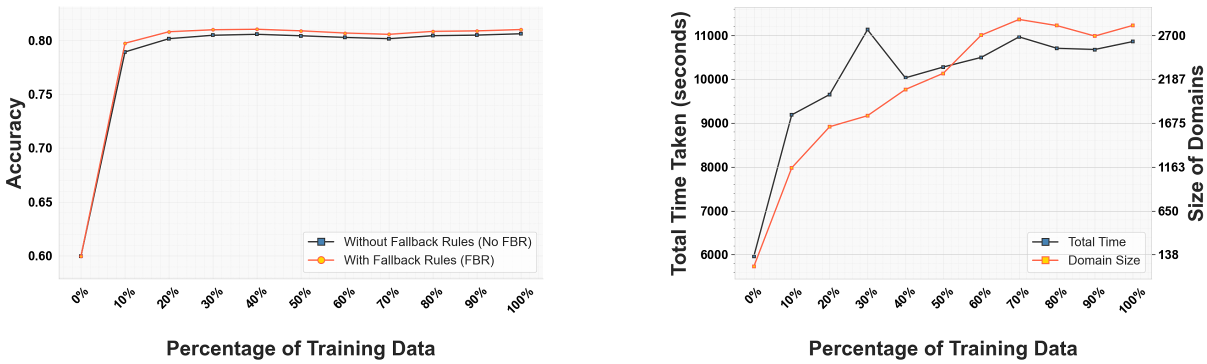
We now report on the results of our experimental evaluation. We use the GQA dataset [3], allowing us to build on the results of [2], which uses the CLEVR [4] synthetic data. Note that we use ground truth ASP representations of the images and queries. We examine our practical heuristic in four different ways. First, we examine the accuracy improvements when employing FAST-DAP. Second, we examine its data efficiency (e.g., how many examples in **EX** are required to provide useful results). Third, we examine the sensitivity of the support threshold for elements of Π_D. Finally, we examine running time. We created our implementation in Python 3.11.7 and use the Clingo solver for the ASP engine. Experiments were run on an Apple M2 machine with a 10-core CPU, and 32GB of RAM. All computations were carried out using only the CPU (the system's GPU was not used). We now present the results of each experiment.

Accuracy. We assess our approach's accuracy against the baseline (no Π^D), evaluating improvements with and without fallback rules (FBR and No FBR), both utilizing FAST-DAP. For the baseline (no FAST-DAP), the ASP solver either provides an answer or returns "empty" if it cannot deduce one. On our test set (disjoint from the examples), the baseline accuracy across all question types was 59.98% without domain information. Incorporating domain information learned from the training set significantly boosted accuracy to 80.62% without fallback rules, and 81.01% with them. To gain deeper insights, we analyze specific question types, a subset of which is presented in Table 1. Some types, such as verification questions, show minimal dependence on domain categorization, while others rely more heavily on it. Additionally, certain questions require translating specific concepts into general terms (FAST-DAP, lines 9-14), like generalizing "banana" to "fruit" or "juice" to "drink." In Table 1, all non-choice queries require such generalization.

Question Type	Baseline	FBR (Ours)	No FBR (Ours)
choose_activity	69.02	95.11	94.84
choose_color	89.80	93.48	93.21
choose_older	0	97.24	97.24
choose_rel	73.88	85.48	81.72
choose_vposition	96.27	94.98	94.93
and	94.25	91.93	91.83
verify_age	86.89	97.54	97.54
verify_color	95.71	96.58	96.44
verify_location	49.28	94.5	94.5
query	36.07	72.83	72.20

Table 1: Evaluation of answering questions. The "Baseline" column shows accuracy (in percentage) without learned domains, "FBR" shows accuracy with learned domains and fallback rules, and "No FBR" shows accuracy with domain atoms but without using fallback rules.

Data Efficiency. In this second experiment, we aimed to find the optimal sample size for learning do-



(a) Accuracy on the test set leveraging learned domains from different training subsets.

(b) Execution time of our algorithm for different sample sizes, run in parallel with identical settings.

Figure 2: Accuracy and running time on different training subsets.

Percentile	Threshold	Accuracy
10	12.3	79.44
20	20.6	79.79
30	30.9	79.89
40	46.4	80.10
50	**59.5**	**80.54**
60	90.8	80.02
70	121.2	79.75

Table 2: Accuracy results on the validation set after removing domains with support below a threshold.

mains. We randomly divided the data as follows: 20% for training, 10% for validation, and the remaining 70% for testing. Instead of using the entire training set at once, we divided it into 11 progressively larger subsets as follows: the first subset served as a baseline model with no samples, the second subset contained 10% of the training data, the third subset included the first 10% plus an additional 10%, making up 20% of the training data, and this pattern continued until the 11th subset, which encompassed all the training data. Each training subset was used independently to learn the domains, and these learned domains were then used to predict the answers in the test set. Figure 2a illustrates the results, showing accuracy across the training data for two scenarios: the black line represents the learned domain without fallback rules, while the red line includes fallback rules. As depicted in Figure 2a, using just 10% of the training set (equivalent to 2% of the entire dataset) achieves a respectable accuracy of 78.93%. With 20% of the training data (4% of the entire dataset), accuracy exceeds 80%. This suggests that a small amount of data can effectively learn domains, with only slight accuracy gains from adding more data.

Threshold Sensitivity. FAST-DAP refines the learned domain set by removing domains whose support falls below a specified threshold. This approach helps regularize the outcome since the domains were derived from the application of possibly noisy data and rules. The threshold is a hyper-parameter determined from the validation set. We used the 10th to 70th percentile support values as potential thresholds. For each, we removed domains with lower support, assessed validation accuracy, and selected the threshold with the highest accuracy. Domains below this final threshold were then removed. Table 2 illustrates the accuracy achieved at different thresholds. Based on this data, we selected a threshold of 59.5, and domains with support below this value were excluded to form the final set of domains.

Running Time. The running time of our algorithm is primarily influenced by the performance of the ASP solver Clingo, and is directly proportional to the number of atoms it processes. Figure 2b illustrates that the running time grows consistently from the base case with no training samples to the scenario where all training samples are used. Incorporating more training samples to learn domains substantially boosts the number of learned new domain atoms, thereby requiring Clingo to process more atoms during deduction. This necessity is the main factor driving the increase in running time. However, note that this increase is bounded by a constant factor related to the domain's size.

5 Conclusion

In this paper, we introduced a practical heuristic algorithm designed to infer domain relationships from a logical representation of data specifically for visual question answering. Our algorithm is highly efficient, requiring just a single pass over the data, and it significantly enhances accuracy compared to using a logical representation that does not leverage domain information. Despite its strong practical performance, an important limitation of our approach is that there are no theoretical guarantees for the solutions it obtains. A promising direction for future research focused on addressing this limitation is to refine our approach by incorporating meta-cognitive AI [8] techniques.

Acknowledgement

This research was funded by Army Research Office (ARO) grant W911NF-24-1-0007.

References

[1] Kinjal Basu, Farhad Shakerin & Gopal Gupta (2020): *AQuA: ASP-Based Visual Question Answering*. In: *PADL*, Springer International Publishing, Cham, pp. 57–72, doi:10.1007/978-3-030-39197-3_4.

[2] Thomas Eiter, Nelson Higuera, Johannes Oetsch & Michael Pritz (2022): *A neuro-symbolic ASP pipeline for visual question answering*. *TPLP* 22(5), pp. 739–754, doi:10.1017/S1471068422000229.

[3] Drew A. Hudson & Christopher D. Manning (2019): *GQA: A New Dataset for Real-World Visual Reasoning and Compositional Question Answering*. In: *CVPR*, pp. 6693–6702, doi:10.1109/CVPR.2019.00686.

[4] 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: *CVPR*, pp. 1988–1997, doi:10.1109/CVPR.2017.215.

[5] Kenneth Marino, Mohammad Rastegari, Ali Farhadi & Roozbeh Mottaghi (2019): *OK-VQA: A Visual Question Answering Benchmark Requiring External Knowledge*. In: *CVPR*, pp. 3190–3199, doi:10.1109/CVPR.2019.00331.

[6] Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino & Roozbeh Mottaghi (2022): *A-OKVQA: A Benchmark for Visual Question Answering Using World Knowledge*. In: *ECCV*, Springer Nature Switzerland, Cham, pp. 146–162, doi:10.1007/978-3-031-20074-8_9.

[7] Peng Wang, Qi Wu, Chunhua Shen, Anthony Dick & Anton van den Hengel (2018): *FVQA: Fact-Based Visual Question Answering*. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40(10), pp. 2413–2427, doi:10.1109/TPAMI.2017.2754246.

[8] Hua Wei, Paulo Shakarian, Christian Lebiere, Bruce Draper, Nikhil Krishnaswamy & Sergei Nirenburg (2024): *Metacognitive AI: Framework and the Case for a Neurosymbolic Approach*, doi:10.48550/arXiv.2406.12147.

Graphical Conditions for the Existence, Unicity and Number of Regular Models

Van-Giang Trinh Belaid Benhamou
LIRICA team, LIS, Aix-Marseille University, Marseille, France
van-giang.trinh@inria.fr belaid.benhamou@lis-lab.fr

Sylvain Soliman François Fages
Inria Saclay, EP Lifeware, Palaiseau, France
Sylvain.Soliman@inria.fr francois.fages@inria.fr

The regular models of a normal logic program are a particular type of partial (i.e. 3-valued) models which correspond to stable partial models with minimal undefinedness. In this paper, we explore graphical conditions on the dependency graph of a finite ground normal logic program to analyze the existence, unicity and number of regular models for the program. We show three main results: 1) a necessary condition for the existence of non-trivial (i.e. non-2-valued) regular models, 2) a sufficient condition for the unicity of regular models, and 3) two upper bounds for the number of regular models based on positive feedback vertex sets. The first two conditions generalize the finite cases of the two existing results obtained by You and Yuan (1994) for normal logic programs with well-founded stratification. The third result is also new to the best of our knowledge. Key to our proofs is a connection that we establish between finite ground normal logic programs and Boolean network theory.

Keywords: logic programming, semantics of negation, canonical model, three-valued model, Datalog, abstract argumentation, Boolean network, feedback vertex set, model counting

1 Introduction

Relating graphical representations of a *normal logic program* (or just *program* if not otherwise said) and its model-theoretic semantics is an interesting research direction in theory that also has many useful applications in practice [14, 6, 20]. Historically, the first studies of this direction focused on the existence of a unique stable model in classes of programs with special graphical properties on (positive) dependency graphs, including positive programs [16], acyclic programs [1], and locally stratified programs [16]. In 1991, Fages gave a simple characterization of stable models as well-supported models in [13], and then showed that for *tight* programs (i.e. without non-well-founded positive justifications), the stable models of the program coincide with the Herbrand models of its Clark's completion [14]. Being finer-represented but more computationally expensive than dependency graphs, several other graphical representations (e.g., cycle and extended dependency graphs, rule graphs, block graphs) were introduced and several improved results were obtained [6, 7, 9, 20]. There are some recent studies on dependency graphs [15, 28], but they still focus only on stable models. In contrast, very few studies were made about regular models despite of their prominent importance in argumentation frameworks [32, 4] and program semantics [18]. The work of [12] showed the unicity of regular and stable models in locally stratified programs. The work of [33] showed two sufficient graphical conditions, one for the coincidence between stable and regular models, and another one for the unicity of regular models. However, these two conditions were only proven in the case of well-founded stratification programs, and the question if they are still valid for any program is still open to date.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 175–187, doi:10.4204/EPTCS.416.16

The stable partial semantics is the 3-valued generalization of the (2-valued) stable model semantics [23]. The regular model semantics not only inherits the advantages of the stable partial model semantics but also imposes two notable principles in non-monotonic reasoning: *minimal undefinedness* and *justifiability* (which is closely related to the concept of labeling-based justification in Doyle's truth maintenance system [10]), making it become one of the well-known semantics in logic programming [33, 18]. Furthermore, regular models in ground programs were proven to correspond to preferred extensions in Dung's frameworks [32] and assumption-based argumentation [4], which are two central focuses in abstract argumentation [3].

Recently, we have proposed a new semantics for finite ground programs, called the *trap space semantics*, which establishes formal links between the model-theoretic and dynamical semantics of a finite ground program [31]. It is built on two newly proposed concepts: *stable* and *supported trap spaces*, which are inspired by the concepts of *trap* (or its duality, *siphon*) in Petri net theory and *trap space* in Boolean network theory [21, 19, 30, 29]. We relate the new semantics to other widely-known semantics, in particular showing that subset-minimal stable trap spaces of a finite ground program coincide with its regular models. Interestingly, the restriction to finite ground programs applies without loss of generality to normal Datalog programs, i.e. normal logic programs built over an alphabet without function symbols, since their Herbrand base and their ground instanciation are finite [5].

Motivated by the above elements, in this paper, we explore graphical conditions on the dependency graph of a finite ground program to analyze the existence of non-trivial (i.e. not 2-valued) regular models and the unicity and multiplicity of regular models for the program. More specifically, we show three main results: 1) the existence of negative cycles is a necessary condition for the existence of non-trivial regular models, 2) the absence of positive cycles is a sufficient condition for the unicity of regular models, and 3) $3^{|U^+|}$ (resp. $2^{|U^+|}$) is an upper bound (resp. a finer upper bound) for the number of regular models in generic (resp. tight) finite ground programs where U^+ is a positive feedback vertex set of the dependency graph. The first two conditions generalize the finite cases of the two existing results obtained by [33] for well-founded stratification normal logic programs. The third result is also new to the best of our knowledge. Key to our proofs is a connection that we establish between finite ground programs and Boolean network theory based on the trap space semantics.

Boolean Networks (BNs) are a simple and efficient mathematical formalism that has been widely applied to many areas from science to engineering [26]. Originated in the early work of [27], studying relationships between the dynamics of a BN and its influence graph has a rich history of research [22, 24]. To date, this research direction is still growing with many prominent and deep results [24, 26, 25]. Hence, the established connection can bring a plenty of existing results in BNs to studying finite ground programs as well as provide a unified framework for exploring and proving more new theoretical results in the logic program theory.

The rest of this paper is organized as follows. In the next section, we recall preliminaries on normal logic programs, regular models, BNs, and related concepts. Section 3 presents the connection that we establish between finite ground programs and BNs. In Section 4, we present the main results on relationships between regular models and graphical conditions. Finally, Section 5 concludes the paper with some perspectives for future work.

2 Preliminaries

We assume that the reader is familiar with the logic program theory and the stable model semantics [16]. Unless specifically stated, a *program* means a normal logic program. In addition, we consider the

Boolean domain $\mathbb{B} = \{\text{true}, \text{false}\} = \{1, 0\}$, and the Boolean connectives used in this paper include $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), $\leftarrow$ (implication), and $\leftrightarrow$ (equivalence).

2.1 Normal logic programs

We consider a first-order language built over an infinite alphabet of variables, and finite alphabets of constant, function and predicates symbol. The set of first-order *terms* is the least set containing variables, constants and closed by application of function symbols. An *atom* is a formula of the form $p(t_1, \dots, t_k)$ where p is a predicate symbol and t_i are terms. A *normal logic program P* is a *finite* set of *rules* of the form

$$p \leftarrow p_1, \dots, p_m, \sim p_{m+1}, \dots, \sim p_k$$

where p and p_i are atoms, $k \geq m \geq 0$, and $\sim$ is a symbol for negation. A fact is a rule with $k = 0$. We denote by $\text{atom}(P)$ the set of atoms appearing in P. For any rule r of the above form, $\text{h}(r) = p$ is the *head* of r, $\text{b}^+(r) = \{p_1, \dots, p_m\}$ is called the *positive body* of r, $\text{b}^-(r) = \{p_{m+1}, \dots, p_k\}$ is called the *negative body* of r, and $\text{bf}(r) = p_1 \wedge \cdots \wedge p_m \wedge \neg p_{m+1} \wedge \cdots \wedge \neg p_k$ is the *body formula* of r. If $\text{b}^-(r) = \emptyset, \forall r \in P$, then P is called a *positive program*. If $\text{b}^+(r) = \emptyset, \forall r \in P$, then P is called a *quasi-interpretation* program.

A term, an atom or a program is *ground* if it contains no variable. The *Herbrand base* is the set of ground atoms formed over the alphabet of the program. It is finite in absence of function symbol, which is the case of *Datalog* programs [5]. The *ground instantiation* of a program P is the set of the ground instances of all rules in P. In the rest of the paper, we restrict ourselves to *finite ground normal logic programs*.

We shall use the fixpoint semantics of normal logic programs [11] to prove many new results in the next sections. To be self-contained, we briefly recall the definition of the *least fixpoint* of a normal logic program P as follows. Let r be the rule $p \leftarrow \sim p_1, \dots, \sim p_k, q_1, \dots, q_j$ and let r_i be rules $q_i \leftarrow \sim q_i^1, \dots, \sim q_i^{l_i}$ where $1 \leq i \leq j$ and $l_i \geq 0$. Then $\sigma_r(\{r_1, \dots, r_j\})$ is the following rule

$$p \leftarrow \sim p_1, \dots, \sim p_k, \sim q_1^1, \dots, \sim q_1^{l_1}, \dots, \sim q_j^1, \dots, \sim q_j^{l_j}.$$

σ_P is the transformation on quasi-interpretation programs: $\sigma_P(Q) = \{\sigma_r(\{r_1, \dots, r_j\}) | r \in P, r_i \in Q, 1 \leq i \leq j\}$. Let $\text{lfp}_i = \sigma_P^i(\emptyset) = \sigma_P(\sigma_P(\dots \sigma_P(\emptyset)))$, then $\text{lfp}(P) = \bigcup_{i \geq 1} \text{lfp}_i$ is the least fixpoint of P. In the case of finite ground programs, $\text{lfp}(P)$ is finite and also a quasi-interpretation finite ground program [11].

2.1.1 Stable and supported partial models

A *3-valued interpretation I* of a finite ground program P is a total function $I\colon \text{atom}(P) \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ that assigns one of the truth values true ($\mathbf{t}$), false ($\mathbf{f}$) or unknown ($\mathbf{u}$), to each atom of P. If $I(a) \neq \mathbf{u}, \forall a \in \text{atom}(P)$, then I is an *Herbrand (2-valued) interpretation* of P. Usually, a 2-valued interpretation is written as the set of atoms that are true in this interpretation. A 3-valued interpretation I characterizes the set of 2-valued interpretations denoted by $\gamma(I)$ as $\gamma(I) = \{J | J \in 2^{\text{atom}(P)}, \forall a \in \text{atom}(P), I(a) \neq \mathbf{u} \Rightarrow J(a) = I(a)\}$. For example, if $I = \{p = \mathbf{t}, q = \mathbf{f}, r = \mathbf{u}\}$, then $\gamma(I) = \{\{p\}, \{p, r\}\}$.

We consider two orders on 3-valued interpretations. The truth order $\leq_t$ is given by $\mathbf{f} <_t \mathbf{u} <_t \mathbf{t}$. Then, $I_1 \leq_t I_2$ iff $I_1(a) \leq_t I_2(a), \forall a \in \text{atom}(P)$. The subset order $\leq_s$ is given by $\mathbf{f} <_s \mathbf{u}$ and $\mathbf{t} <_s \mathbf{u}$. Then, $I_1 \leq_s I_2$ iff $I_1(a) \leq_s I_2(a), \forall a \in \text{atom}(P)$. In addition, $I_1 \leq_s I_2$ iff $\gamma(I_1) \subseteq \gamma(I_2)$, i.e., $\leq_s$ is identical to the subset partial order.

Let f be a propositional formula on $\mathrm{atom}(P)$. Then the valuation of f under a 3-valued interpretation I (denoted by $I(f)$) is defined recursively as follows:

$$I(f) = \begin{cases} I(a) & \text{if } f = a, a \in \mathrm{atom}(P) \\ \neg I(f_1) & \text{if } f = \neg f_1 \\ \min_{\leq_t}(I(f_1), I(f_2)) & \text{if } f = f_1 \wedge f_2 \\ \max_{\leq_t}(I(f_1), I(f_2)) & \text{if } f = f_1 \vee f_2 \end{cases}$$

where $\neg\mathbf{t} = \mathbf{f}, \neg\mathbf{f} = \mathbf{t}, \neg\mathbf{u} = \mathbf{u}$, 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 3-valued interpretation I is a *3-valued model* of a finite ground program P iff for each rule $r \in P$, $I(\mathrm{bf}(r)) \leq_t I(\mathrm{h}(r))$.

Definition 1. *Let I be a 3-valued interpretation of P. We build the* reduct P^I *as follows.*

- *Remove any rule $a \leftarrow a_1, \ldots, a_m, \sim b_1, \ldots, \sim b_k \in P$ if $I(b_i) = \boldsymbol{t}$ for some $1 \leq i \leq k$.*
- *Afterwards, remove any occurrence of $\sim b_i$ from P such that $I(b_i) = \boldsymbol{f}$.*
- *Then, replace any occurrence of $\sim b_i$ left by a special atom $\boldsymbol{u}$ ($\boldsymbol{u} \notin \mathrm{atom}(P)$).*

P^I is positive and has a unique $\leq_t$-least 3-valued model. See [23] for the method for computing this $\leq_t$-least 3-valued model. Then I is a stable partial model *of P iff I is equal to the $\leq_t$-least 3-valued model of P^I. A stable partial model I is a regular model if it is $\leq_s$-minimal. A regular model is non-trivial if it is not 2-valued.*

The *Clark's completion* of a finite ground program P (denoted by $\mathrm{cf}(P)$) consists of the following sentences: for each $p \in \mathrm{atom}(P)$, let $r_1, \ldots, r_k$ be all the rules of P having the same head p, then $p \leftrightarrow \mathrm{bf}(r_1) \vee \cdots \vee \mathrm{bf}(r_k)$ is in $\mathrm{cf}(P)$. If there is no rule whose head is p, then the equivalence is $p \leftrightarrow \mathbf{f}$. Let $\mathrm{rhs}(a)$ denote the right hand side of atom a in $\mathrm{cf}(P)$. A 3-valued interpretation I is a 3-valued model of $\mathrm{cf}(P)$ iff for every $a \in \mathrm{atom}(P)$, $I(a) = I(\mathrm{rhs}(a))$. We define a *supported partial model* of P as a 3-valued model of $\mathrm{cf}(P)$. Note that 2-valued stable (resp. supported) partial models are stable (resp. supported) models.

2.1.2 Dependency and transition graphs

The Dependency Graph (DG) of a finite ground program P (denoted by $\mathrm{dg}(P)$) is a signed directed graph (V, E) on the set of signs $\{\oplus, \ominus\}$ where $V = \mathrm{atom}(P)$ and $(uv, \oplus) \in E$ (resp. $(uv, \ominus) \in E$) iff there is a rule $r \in P$ such that $v = \mathrm{h}(r)$ and $u \in \mathrm{b}^+(r)$ (resp. $u \in \mathrm{b}^-(r)$). An arc $(uv, \oplus)$ is positive, whereas an arc $(uv, \ominus)$ is negative. Since $\mathrm{atom}(P)$ is finite, the DG of P is a finite graph, thus we can apply the finite graph theory. A cycle of $\mathrm{dg}(P)$ is positive (resp. negative) if it contains an even (resp. odd) number of negative arcs. A positive (resp. negative) feedback vertex set is a set of vertices that intersect all positive (resp. negative) cycles of $\mathrm{dg}(P)$. The positive DG of P (denoted by $\mathrm{dg}^+(P)$) is a sub-graph of $\mathrm{dg}(P)$ that has the same set of vertices but contains only positive arcs. P is *locally stratified* if every cycle of $\mathrm{dg}(P)$ contains no negative arc [16]. P is *tight* if $\mathrm{dg}^+(P)$ has no cycle [14]. P is *well-founded stratification* if there is a topological order on the set of Strongly Connected Components (SCCs) of $\mathrm{dg}(P)$ and for every SCC B, there exists SCC $A \leq B$ and for any SCC C, if $C \leq A$ then there are only positive arcs from atoms in C to atoms in A [33]. Herein, $A \leq B$ iff there is a path from some atom in A to some atom in B. In the case of finite ground programs, the above definition of well-founded stratification (which was orginally defined for both finite and infinite ground programs) is equivalent to that a finite ground program is well-founded stratification iff there is a topological order of its dependency graph such that every SCC at the lowest level only contains positive arcs.

The *immediate consequence operator* (or the *T_P operator*) is defined as a mapping $T_P\colon 2^{\mathrm{atom}(P)} \to 2^{\mathrm{atom}(P)}$ such that $T_P(I)(a) = I(\mathrm{rhs}(a))$ where I is a 2-valued interpretation. If I is a 2-valued interpretation, then P^I is exactly the reduct defined in [16] and the unique $\leq_t$-least model of P^I is 2-valued. The *Gelfond-Lifschitz operator* (or the *F_P operator*) is defined as a mapping $F_P\colon 2^{\mathrm{atom}(P)} \to 2^{\mathrm{atom}(P)}$ such that $F_P(I)$ is the unique $\leq_t$-least model of P^I [16]. The *stable* (resp. *supported*) *transition graph* of P is a directed graph (denoted by $\mathrm{tg}_{st}(P)$ (resp. $\mathrm{tg}_{sp}(P)$)) on the set of all possible 2-valued interpretations of P such that (I,J) is an arc of $\mathrm{tg}_{st}(P)$ (resp. $\mathrm{tg}_{sp}(P)$) iff $J = F_P(I)$ (resp. $J = T_P(I)$). A *trap domain* of a directed graph is a set of vertices having no out-going arcs.

2.1.3 Stable and supported trap spaces

In [31], we introduce a new semantics for finite ground programs, called the *trap space semantics*. This semantics shall be used in this work as the bridge between finite ground programs and Boolean networks. To be self-contained, we briefly recall the definition and essential properties of this semantics.

A set S of 2-valued interpretations of a finite ground program P is called a *stable trap set* (resp. *supported trap set*) of P if $\{F_P(I) | I \in S\} \subseteq S$ (resp. $\{T_P(I) | I \in S\} \subseteq S$). A 3-valued interpretation I of a finite ground program P is called a *stable trap space* (resp. *supported trap space*) of P if $\gamma(I)$ is a stable (resp. supported) trap set of P. By definition, a stable (resp. supported) trap set of P is a trap domain of $\mathrm{tg}_{st}(P)$ (resp. $\mathrm{tg}_{sp}(P)$). Hence, we can deduce that a 3-valued interpretation I is a stable (resp. supported) trap space of P if $\gamma(I)$ is a trap domain of $\mathrm{tg}_{st}(P)$ (resp. $\mathrm{tg}_{sp}(P)$). We also show in [31] that I is a supported trap space of P iff I is 3-valued model of $\overleftarrow{\mathrm{cf}}(P)$ w.r.t. to the order $\leq_s$ where $\overleftarrow{\mathrm{cf}}(P)$ is the $\leftarrow$ part of the Clark's completion of P, and a stable (resp. supported) partial model of P is also a stable (resp. supported) trap space of P.

Example 1. *Consider finite ground program P_1 (taken from [17]) where $P_1 = \{p \leftarrow \sim q; q \leftarrow \sim p; r \leftarrow q\}$. Herein, we use ';' to separate program rules. Figures 1 (a), (b), and (c) show the dependency graph, the stable transition graph, and the supported transition graph of P_1, respectively. P_1 is tight, but neither locally stratified nor well-founded stratification. P_1 has five stable (also supported) trap spaces: $I_1 = \{p = \boldsymbol{t}, q = \boldsymbol{f}, r = \boldsymbol{u}\}$, $I_2 = \{p = \boldsymbol{f}, q = \boldsymbol{t}, r = \boldsymbol{u}\}$, $I_3 = \{p = \boldsymbol{u}, q = \boldsymbol{u}, r = \boldsymbol{u}\}$, $I_4 = \{p = \boldsymbol{t}, q = \boldsymbol{f}, r = \boldsymbol{f}\}$, and $I_5 = \{p = \boldsymbol{f}, q = \boldsymbol{t}, r = \boldsymbol{t}\}$. Among them, only I_3, I_4, and I_5 are stable (also supported) partial models of P_1. P_1 has two regular models (I_4 and I_5). The least fixpoint of P_1 is $lfp(P_1) = \{p \leftarrow \sim q; q \leftarrow \sim p; r \leftarrow \sim p\}$.*

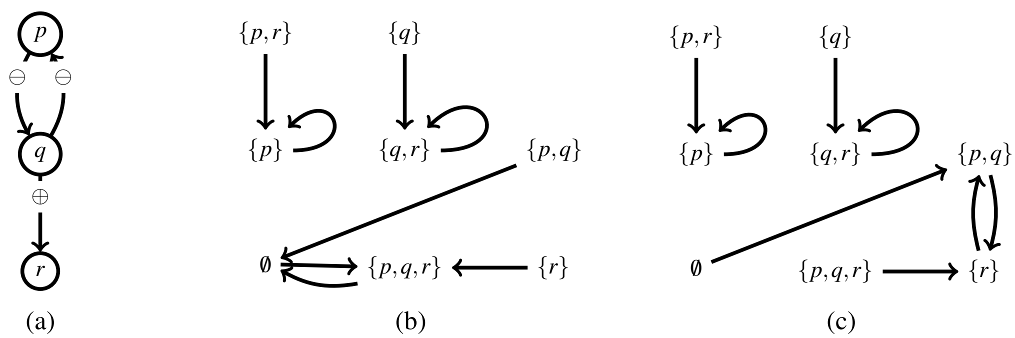


Figure 1: (a) $\mathrm{dg}(P_1)$, (b) $\mathrm{tg}_{st}(P_1)$, and (c) $\mathrm{tg}_{sp}(P_1)$.

2.2 Boolean networks

A Boolean Network (BN) f is a *finite* set of Boolean functions on a finite set of Boolean variables denoted by var_f. Each variable v is associated with a Boolean function $f_v \colon \mathbb{B}^{|\text{var}_f|} \rightarrow \mathbb{B}$. f_v is called *constant* if it is always either 0 or 1 regardless of its arguments. A state s of f is a mapping $s\colon \text{var}_f \mapsto \mathbb{B}$ that assigns either 0 (inactive) or 1 (active) to each variable. We can write s_v instead of $s(v)$ for short.

Let x be a state of f. We use $x[v \leftarrow a]$ to denote the state y so that $y_v = a$ and $y_u = x_u, \forall u \in \text{var}_f, u \neq v$ where $a \in \mathbb{B}$. The Influence Graph (IG) of f (denoted by $\text{ig}(f)$) is a signed directed graph (V,E) on the set of signs $\{\oplus, \ominus\}$ where $V = \text{var}_f$, $(uv, \oplus) \in E$ (i.e., u positively affects the value of f_v) iff there is a state x such that $f_v(x[u \leftarrow 0]) < f_v(x[u \leftarrow 1])$, and $(uv, \ominus) \in E$ (i.e., u negatively affects the value of f_v) iff there is a state x such that $f_v(x[u \leftarrow 0]) > f_v(x[u \leftarrow 1])$.

At each time step t, variable v can update its state to $s'(v) = f_v(s)$, where s (resp. s') is the state of f at time t (resp. $t+1$). An *update scheme* of a BN refers to how variables update their states over (discrete) time [26]. Various update schemes exist, but the primary types are *synchronous*, where all variables update simultaneously, and *fully asynchronous*, where a single variable is non-deterministically chosen for updating. By adhering to the update scheme, the BN transitions from one state to another, which may or may not be the same. This transition is referred to as the *state transition*. Then the dynamics of the BN is captured by a directed graph referred to as the State Transition Graph (STG). We use $\text{sstg}(f)$ (resp. $\text{astg}(f)$) to denote the synchronous (resp. asynchronous) STG of f.

A non-empty set of states is a *trap set* if it has no out-going arcs on the STG of f. An *attractor* is a subset-minimal trap set. An attractor of size 1 (resp. at least 2) is called a fixed point (resp. cyclic attractor). A *sub-space* m of a BN is a mapping $m\colon \text{var}_f \mapsto \mathbb{B} \cup \{\star\}$. A sub-space m is equivalent to the set of all states s such that $s(v) = m(v), \forall v \in \text{var}_f, m(v) \neq \star$. With abuse of notation, we use m and its equivalent set of states interchangeably. For example, $m = \{v_1 = \star, v_2 = 1, v_3 = 1\} = \{011, 111\}$ (for simplicity, we write states as a sequence of values). If a sub-space is also a trap set, it is a *trap space*. Unlike trap sets and attractors, trap spaces of a BN are independent of the update scheme [19]. Then a trap space m is minimal iff there is no other trap space m' such that $m' \subset m$. It is easy to derive that a minimal trap space contains at least one attractor of the BN regardless of the update scheme.

Example 2. *Consider BN f_1 with $f_p = \neg q, f_q = \neg p, f_r = q$. Figures 2 (a), (b), and (c) show the influence graph, the synchronous STG, and the asynchronous STG of f_1. Attractor states are highlighted with boxes. $\text{sstg}(f_1)$ has two fixed points and one cyclic attractor, whereas $\text{astg}(f_1)$ has only two fixed points. f_1 has five trap spaces: $m_1 = 10\star$, $m_2 = 01\star$, $m_3 = \star\star\star$, $m_4 = 100$, and $m_5 = 011$. Among them, m_4 and m_5 are minimal.*

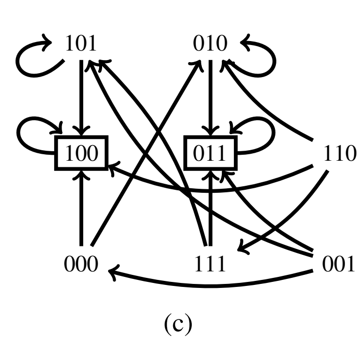


Figure 2: (a) $\text{ig}(f_1)$, (b) $\text{sstg}(f_1)$, and (c) $\text{astg}(f_1)$.

3 Finite ground normal logic programs and Boolean networks

We define a BN encoding for finite ground programs in Definition 2. Then, we show two relationships between a finite ground program and its encoded BN (see Theorems 1 and 3).

Definition 2. *Let P be a finite ground program. We define a BN f encoding P as follows: $var_f = atom(P)$, $f_v = \bigvee_{r \in P, v = h(r)} bf(r), \forall v \in var_f$. Conventionally, if there is no rule $r \in P$ such that $h(r) = v$, then $f_v = 0$. By considering 1 (resp. 0) as **t** (resp. **f**), and $\star$ as **u**, sub-spaces (resp. states) of f are identical to 3-valued (resp. 2-valued) interpretations of P.*

Theorem 1. *Let P be a finite ground program and f be its encoded BN. Then $ig(f) \subseteq dg(P)$.*

Proof. By construction, $\mathrm{ig}(f)$ and $\mathrm{dg}(P)$ have the same set of vertices. Let $\mathrm{in}^+_f(v)$ (resp. $\mathrm{in}^+_P(v)$) denote the set of vertices u such that $(uv, \oplus)$ is an arc of $\mathrm{ig}(f)$ (resp. $\mathrm{dg}(P)$). We define $\mathrm{in}^-_f(v)$ (resp. $\mathrm{in}^-_P(v)$) similarly. We show that $\mathrm{in}^+_f(v) \subseteq \mathrm{in}^+_P(v)$ and $\mathrm{in}^-_f(v) \subseteq \mathrm{in}^-_P(v)$ for every $v \in \mathrm{atom}(P)$ (*). Consider atom u. The case that both u and $\sim u$ appear in rules whose heads are v is trivial. For the case that only u appears in rules whose heads are v, u is essential in f_v by construction, and it positively affects the value of f_v, leading to $u \in \mathrm{in}^+_f(v)$ and $u \not\in \mathrm{in}^-_f(v)$. This implies that (*) still holds. The case that only $\sim u$ appears in rules whose heads are v is similar. By (*), we can conclude that $\mathrm{ig}(f) \subseteq \mathrm{dg}(P)$, i.e., $\mathrm{ig}(f)$ is a sub-graph of $\mathrm{dg}(P)$. In addition, if P is a quasi-interpretation finite ground program, then $\mathrm{ig}(f) = \mathrm{dg}(P)$. ☐

Lemma 2 (derived from Theorem 4.5 of [17])**.** *Let P be a finite ground program and f be its encoded BN. Then $tg_{sp}(P) = sstg(f)$.*

Theorem 3. *Let P be a finite ground program and f be its encoded BN. Then supported trap spaces of P coincide with trap spaces of f.*

Proof. By Lemma 2, $\mathrm{tg}_{sp}(P) = \mathrm{sstg}(f)$. Note that trap spaces of f are the same under both the synchronous and asynchronous update schemes [19]. Hence, trap spaces of f coincide with trap spaces of $\mathrm{sstg}(f)$. Since $\mathrm{tg}_{sp}(P) = \mathrm{sstg}(f)$, supported trap spaces of P coincide with trap spaces of f. ☐

For illustration, BN f_1 of Example 2 is the encoded BN of finite ground program P_1 of Example 1. $\mathrm{tg}_{sp}(P_1)$ is identical to $\mathrm{sstg}(f_1)$, and the five supported trap spaces of P_1 are identical to the five trap spaces of f_1. In addition, P_1 is tight and $\mathrm{ig}(f_1) = \mathrm{dg}(P_1)$.

4 Graphical analysis results

In this section, we present our new results on graphical conditions for several properties of regular models in finite ground normal logic programs by exploiting the connection established in Section 3.

4.1 Preparations

For convenience, we first recall several existing results in both logic programs and Boolean networks that shall be used later.

Theorem 4 ([17])**.** *Let P be a quasi-interpretation finite ground program. Then $tg_{st}(P) = tg_{sp}(P)$, i.e., the stable and supported transition graphs of P are the same.*

Theorem 5 ([17])**.** *Let P be a finite ground program and $lfp(P)$ denote its least fixpoint. Then P and $lfp(P)$ have the same stable transition graph.*

Theorem 6 (Theorem 6 of [28]). *Let P be a finite ground program and lfp(P) denote its least fixpoint. If P is locally stratified, then dg(lfp(P)) has no cycle.*

Lemma 7. *Let P be a finite ground program and lfp(P) denote its least fixpoint. If dg(P) is has no negative cycle, then dg(lfp(P)) has no negative cycle.*

Proof. It directly follows from Lemma 5.3 of [14]. □

Proposition 8 ([31]). *Let P be a finite ground program. Let $T(P)$ denote the set of all supported trap spaces of P. Let $C(P)$ denote the set of all 3-valued models of cf(P) (i.e., the Clark's completion of P). For every supported trap space $I \in T(P)$, there is a model $I' \in C(P)$ such that $I' \leq_s I$.*

Sketch of proof. Let I^j be an arbitrary supported trap space in $T(P)$. We construct a 3-valued interpretation I^{j+1} as follows: $\forall a \in \mathrm{atom}(P), I^{j+1}(a) = I^j(\mathrm{rhs}(a))$. We prove that I^{j+1} is also a supported trap space of P. For every supported trap space I in $T(P)$, we start with $I^j = I$ and repeat the above process by increasing j by 1, and finally reach the case that $I^{j+1} = I^j$ because $\gamma(I)$ is finite. By construction, $I^j(a) = I^j(\mathrm{rhs}(a)), \forall a \in \mathrm{atom}(P)$, and $I^j \leq_s I$. Hence, by setting $I' = I^j$, there is a model $I' \in C(P)$ such that $I' \leq_s I$. □

Theorem 9 ([31]). *Let P be a finite ground program. Then a 3-valued interpretation I is a regular model of P iff I is a $\leq_s$-minimal stable trap space of P.*

Sketch of proof. Let lfp(P) be the least fixpoint of P. By Proposition 8, we can deduce that $\leq_s$-minimal supported trap spaces of lfp(P) coincide with $\leq_s$-minimal supported (also stable) partial models spaces of lfp(P). P and lfp(P) have the same set of stable partial models [2]. By Theorem 5, P and lfp(P) have the same stable transition graph, thus they have the same set of stable trap spaces. Since stable trap spaces of lfp(P) coincide with its supported trap spaces, we can conclude the theorem. □

Theorem 10 (Theorem 1 of [24]). *Let f be a BN. If ig(f) has no cycle, astg(f) has a unique attractor that is also the unique fixed point of f.*

Theorem 11 (Theorem 12 of [24]). *Let f be a BN. If ig(f) has no negative cycle, then astg(f) has no cyclic attractor.*

4.2 Unicity of regular and stable models

To illustrate better applications of the connection between finite ground programs and Boolean networks, we start with providing a probably simpler proof for the finite case of a well-known result on the unicity of regular and stable models in locally stratified programs [12].

Theorem 12 ([12]). *If P is a locally stratified finite ground program, then P has a unique regular model that is also the unique stable model of P.*

New proof. Let lfp(P) denote the least fixpoint of P. Let f be the encoded BN of lfp(P). By Theorem 6, dg(lfp(P)) has no cycle. Since ig(f) is a sub-graph of dg(lfp(P)) by Theorem 1, it also has no cycle. By Theorem 10, astg(f) has a unique attractor that is also the unique fixed point of f. P and lfp(P) have the same set of regular (also stable) models [2]. By Theorem 9, regular models of lfp(P) are $\leq_s$-minimal stable trap spaces of lfp(P). Since lfp(P) is a quasi-interpretation finite ground program, its stable trap spaces coincide with its supported trap spaces. Supported trap spaces of lfp(P) coincide with trap spaces of f by Theorem 3. Hence, regular models of P coincide with $\leq_s$-minimal trap spaces of f. Since the number of $\leq_s$-minimal trap spaces of f are a lower bound of the number of attractors of $\mathrm{astg}(f)$ and f has at least one $\leq_s$-minimal trap space [19], f has a unique $\leq_s$-minimal trap space that is also the unique fixed point of f. Hence, P has a unique regular model that is also the unique stable model of P. □

4.3 Existence of non-trivial regular models

Theorem 13 (Theorem 5.3(i) of [33])**.** *Let P be a well-founded stratification normal logic program. If $dg(P)$ has no negative cycle, then all the regular models of P are 2-valued.*

Theorem 13 provides a sufficient (resp. necessary) condition on the dependency graph for the non-existence (resp. existence) of non-trivial regular models, but it is only limited to well-founded stratification normal logic programs. Note that the well-founded stratification of a normal logic program is defined based on the ground instantiation of this program [33], and the set of all possible well-founded stratification programs in the finite case is only a small piece of the set of all possible finite ground programs [33]. To the best of our knowledge, the question if it is valid for any finite ground program is still open to date. We answer this question in Theorem 14.

Theorem 14. *Let P be a finite ground program. If $dg(P)$ has no negative cycle, then all the regular models of P are 2-valued.*

Proof. Let $\mathrm{lfp}(P)$ be the least fixpoint of P. By Lemma 7, $\mathrm{dg}(\mathrm{lfp}(P))$ has no negative cycle. Let f be the encoded BN of $\mathrm{lfp}(P)$. Since $\mathrm{ig}(f)$ is a sub-graph of $\mathrm{dg}(\mathrm{lfp}(P))$ by Theorem 1, $\mathrm{ig}(f)$ also has no negative cycle. By Theorem 11, $\mathrm{astg}(f)$ (i.e., the asynchronous transition graph of f) has no cyclic attractor. This implies that all attractors of $\mathrm{astg}(f)$ are fixed points (*). Assume that f has a $\leq_s$-minimal trap space (say m) that is not a fixed point. Since every $\leq_s$-minimal trap space of f contains at least one attractor of $\mathrm{astg}(f)$ [19], there is an attractor (say A) of $\mathrm{astg}(f)$ such that $A \subseteq \gamma(m)$. By (*), A is a fixed point, leading to $A <_s m$. This is a contradiction because m is $\leq_s$-minimal. Hence, all $\leq_s$-minimal trap spaces of f are fixed points.

By Theorem 3, trap spaces of f coincide with supported trap spaces of $\mathrm{lfp}(P)$. $\mathrm{lfp}(P)$ is a quasi-interpretation finite ground program, thus $\mathrm{tg}_{st}(\mathrm{lfp}(P)) = \mathrm{tg}_{sp}(\mathrm{lfp}(P))$. It follows that its supported trap spaces are also its stable trap spaces. Hence, $\leq_s$-minimal trap spaces of f are $\leq_s$-minimal stable trap spaces of $\mathrm{lfp}(P)$. This implies that all $\leq_s$-minimal stable trap spaces of $\mathrm{lfp}(P)$ are 2-valued. By Theorem 9, all regular models of $\mathrm{lfp}(P)$ are 2-valued. P and $\mathrm{lfp}(P)$ have the same set of regular models [2]. Hence, all regular models of P are 2-valued. □

Theorem 14 implies that the undefinedness is only needed if there is a negative cycle in the DG, i.e., the regular model and stable model semantics are the same under the absence of negative cycles. In addition, we can get from Theorem 14 a straightforward corollary: if the DG of a finite ground program has no negative cycle, then it has at least one stable model. The reason is because a finite ground program always has at least one regular model [33]. This corollary is exactly the generalization of the finite case of Theorem 5.7 of [33] for well-founded stratification programs.

4.4 Unicity of regular models

The work of [33] shows a sufficient condition for the unicity of regular models for well-founded stratification normal logic programs.

Theorem 15 (Theorem 5.3(ii) of [33])**.** *Let P be a well-founded stratification program. If $dg(P)$ has no positive cycle, P has a unique regular model.*

Hereafter, we would like to show that the finite case of Theorem 15 is also true for any finite ground program. Note however that the technique of using least fixpoint applied for negative cycles seems difficult to use for positive cycles because there is some finite ground program whose dependency graph has no positive cycle but the dependency graph of its least fixpoint can have positive cycle (e.g., $P = \{a \leftarrow c; b \leftarrow c; c \leftarrow \sim a, \sim b\}$). We here use another approach.

Theorem 16 (Theorem 3.4 of [22])**.** *Let f be a BN. If $ig(f)$ has no positive cycle, then $astg(f)$ has a unique attractor.*

Theorem 17 (Lemma 16 of [8])**.** *Supported partial models of a tight finite ground program coincide with its stable partial models.*

Lemma 18. *Let P be a finite ground program and f be its encoded BN. If P is tight, then regular models of P coincide with $\leq_s$-minimal trap spaces of f.*

Proof. Since P is tight, stable partial models of P coincide with supported partial models of P (i.e., 3-valued models of $\mathrm{cf}(P)$) by Theorem 17. Then regular models of P coincide with $\leq_s$-minimal supported partial models of P. We have that trap spaces of f coincide with supported trap spaces of P by Theorem 3. By Proposition 8, $\leq_s$-minimal supported partial models of P coincide with $\leq_s$-minimal supported trap spaces of P. Hence, regular models of P coincide with $\leq_s$-minimal trap spaces of f. □

Theorem 19. *Let P be a finite ground program. If $dg(P)$ has no positive cycle, then P has a unique regular model.*

Proof. Since $\mathrm{dg}(P)$ has no positive cycle, $\mathrm{dg}^+(P)$ (i.e., the positive dependency graph of P) has no cycle, i.e., P is tight. Let f be the encoded BN of P. By Lemma 18, regular models of P coincide with $\leq_s$-minimal trap spaces of f. Since $\mathrm{ig}(f)$ is a sub-graph of $\mathrm{dg}(P)$, it also has no positive cycle. By Theorem 16, $\mathrm{astg}(f)$ has a unique attractor. Since every $\leq_s$-minimal trap space of f contains at least one attractor of $\mathrm{astg}(f)$ and f has at least one $\leq_s$-minimal trap space [19], f has a unique $\leq_s$-minimal trap space. Hence, we can conclude that P has a unique regular model. □

Since a stable model is also a regular model, Theorem 19 implies that if $\mathrm{dg}(P)$ has no positive cycle, then P has at most one stable model. In addition, P may have no stable model because the unique regular model may be not 2-valued. This result seems to be already known in the folklore of the logic program theory, but to the best of our knowledge, there is no existing formal proof for it except the one that we have directly proved recently in [28].

4.5 Upper bound for number of regular models

To the best of our knowledge, there is no existing work connecting between regular models of a finite ground program and (positive/negative) feedback vertex sets of its dependency graph. In [28], we have shown that $2^{|U^+|}$ is an upper bound for the number of stable models where U^+ is a positive feedback vertex set of the dependency graph. Since stable models are 2-valued regular models, we can naturally generalize this result for the case of regular models, i.e., $3^{|U^+|}$ is an upper bound for the number of regular models. The underlying intuition for the base of three is that in a regular model, the value of an atom can be **t**, **f**, or **u**.

Theorem 20. *Let P be a finite ground program. Let U^+ be a positive feedback vertex set of $dg(P)$. Then the number of regular models of P is at most $3^{|U^+|}$.*

Proof. By Theorem 9, regular models of P coincide with $\leq_s$-minimal stable trap spaces of P. For any mapping $\widehat{I} : U^+ \to \{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$, we build a new finite ground program $\widehat{P}$ from P as follows. First, remove from P all the rules whose heads belong to U^+. Second, remove all the rules whose body formulas are false under the values of the atoms in U^+ and otherwise remove all the appearances of the atoms that are in U^+ and not assigned to $\mathbf{u}$ in $\widehat{I}$. Third, for any atom $a \in U^+$ such that $\widehat{I}(a) = \mathbf{u}$, add the rule $a \leftarrow \sim a$. We can see that the part of $\mathrm{tg}_{st}(P)$ induced by $\widehat{I}$ is isomorphic to $\mathrm{tg}_{st}(\widehat{P})$. Hence, $\leq_s$-minimal stable trap spaces of P induced by $\widehat{I}$ one-to-one correspond to those of $\widehat{P}$. U^+ intersects all positive cycles of $\mathrm{dg}(P)$. Every atom $a \in U^+$ such that $\widehat{I}(a) \neq \mathbf{u}$ is removed from $\mathrm{dg}(P)$. In the case that $a \in U^+$ and $\widehat{I}(a) = \mathbf{u}$, all the arcs ending at a are removed and an negative arc $(aa, \ominus)$ is added. It follows that $\mathrm{dg}(\widehat{P})$ has no positive cycle. By Theorem 19, $\widehat{P}$ has a unique $\leq_s$-minimal stable trap space. There are $3^{|U^+|}$ possible mappings $\widehat{I}$, thus we can conclude the theorem. □

Theorem 21 (Theorem 3.5 of [22]). *Let f be a BN. Let U^+ be a positive feedback vertex set of $ig(f)$. Then the number of attractors of $astg(f)$ is at most $2^{|U^+|}$.*

We observed that the bound of $3^{|U^+|}$ is too rough for many example finite ground programs in the literature. Then inspired by Theorem 21 for an upper bound for the number of attractors of an asynchronous BN, we obtain an interesting result for tight finite ground programs.

Theorem 22. *Let P be a tight finite ground program. Let U^+ be a positive feedback vertex set of $dg(P)$. Then the number of regular models of P is at most $2^{|U^+|}$.*

Proof. Let f be the encoded BN of P. By Lemma 18, regular models of P coincide with $\leq_s$-minimal trap spaces of f. By definition, U^+ intersects all positive cycles of $\mathrm{dg}(P)$. Since $\mathrm{ig}(f)$ is a sub-graph of $\mathrm{dg}(P)$, every positive cycle of $\mathrm{ig}(f)$ is also a positive cycle of $\mathrm{dg}(P)$. Hence, U^+ is also a positive feedback vertex set of $\mathrm{ig}(f)$. By Theorem 21, the number of attractors of $\mathrm{astg}(f)$ is at most $2^{|U^+|}$. Since the number of $\leq_s$-minimal trap spaces of f is a lower bound of the number of attractors of $\mathrm{astg}(f)$ [19], the number of regular models of P is at most $2^{|U^+|}$. □

5 Conclusion and perspectives

In this paper, we have shown three main results relating some graphical properties of a finite ground normal logic program to the set of its regular models, namely 1) the presence of negative cycles as a necessary condition for the existence of non-trivial regular models, 2) the absence of positive cycles as a sufficient condition for the unicity of regular models, and 3) two upper bounds on the number of regular models for, respectively generic and tight, finite ground normal logic programs based on the size of positive feedback vertex sets in their dependency graph. The first two conditions generalize the finite cases of the two existing results obtained by [33] for well-founded stratification normal logic programs. Our proofs use an encoding of finite ground normal logic programs by Boolean networks, the equivalence established between regular models and minimal trap spaces, and some recent results obtained in Boolean network theory.

We believe that the established connection can provide more results for the study of Datalog programs and abstract argumentation, and might also be worth considering for normal logic programs without finiteness assumption on their ground intantiation. The results presented in this paper use conditions on either positive cycles or negative cycles. It is thus natural to think that by using both kinds of cycles simultaneously, improved results might be obtained. Finally, we also conjecture that the upper bound for tight finite ground normal logic programs presented here, is in fact valid for generic ones.

Acknowledgments

This work was supported by Institut Carnot STAR, Marseille, France.

References

[1] Krzysztof R Apt & Marc Bezem (1991): *Acyclic programs*. New Gener. Comput. 9, pp. 335–363, doi:10.1007/BF03037168.

[2] 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.

[3] Pietro Baroni, Martin Caminada & Massimiliano Giacomin (2011): *An introduction to argumentation semantics*. Knowl. Eng. Rev. 26(4), pp. 365–410, doi:10.1017/S0269888911000166.

[4] 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.

[5] Stefano Ceri, Georg Gottlob & Letizia Tanca (1990): *Logic Programming and Databases: An Overview*. Springer, doi:10.1007/978-3-642-83952-8.

[6] Stefania Costantini (2006): *On the existence of stable models of non-stratified logic programs*. Theory Pract. Log. Program. 6(1-2), pp. 169–212, doi:10.1017/S1471068405002589.

[7] Stefania Costantini & Alessandro Provetti (2011): *Conflict, Consistency and Truth-Dependencies in Graph Representations of Answer Set Logic Programs*. In: Second International Workshop on Graph Structures for Knowledge Representation and Reasoning, Springer, pp. 68–90, doi:10.1007/978-3-642-29449-5_3.

[8] 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.

[9] Yannis Dimopoulos & Alberto Torres (1996): *Graph Theoretical Structures in Logic Programs and Default Theories*. Theor. Comput. Sci. 170(1-2), pp. 209–244, doi:10.1016/S0304-3975(96)80707-9.

[10] Jon Doyle (1979): *A Truth Maintenance System*. Artif. Intell. 12(3), pp. 231–272, doi:10.1016/0004-3702(79)90008-0.

[11] Phan Minh Dung & Kanchana Kanchanasut (1989): *A Fixpoint Approach to Declarative Semantics of Logic Programs.* In: Proc. of NACLP, MIT Press, pp. 604–625.

[12] 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.

[13] 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.

[14] Francois Fages (1994): *Consistency of Clark's completion and existence of stable models*. Methods Log. Comput. Sci. 1(1), pp. 51–60. Available at https://lifeware.inria.fr/~fages/Papers/MLCS.pdf.

[15] Jorge Fandinno & Vladimir Lifschitz (2023): *Positive Dependency Graphs Revisited*. Theory Pract. Log. Program. 23(5), pp. 1128–1137, doi:10.1017/S1471068422000333.

[16] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In: Proc. of ICLP, MIT Press, pp. 1070–1080.

[17] Katsumi Inoue & Chiaki Sakama (2012): *Oscillating Behavior of Logic Programs*. In: Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz, Springer, pp. 345–362, doi:10.1007/978-3-642-30743-0_23.

[18] Tomi Janhunen, Ilkka Niemelä, Dietmar Seipel, Patrik Simons & Jia-Huai You (2006): *Unfolding partiality and disjunctions in stable model semantics*. ACM Trans. Comput. Log. 7(1), pp. 1–37, doi:10.1145/1119439.1119440.

[19] 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.

[20] Thomas Linke (2001): *Graph Theoretical Characterization and Computation of Answer Sets*. In: Proc. of IJCAI, Morgan Kaufmann, pp. 641–648.

[21] T. Murata (1989): *Petri nets: Properties, analysis and applications*. Proc. IEEE 77(4), pp. 541–580, doi:10.1109/5.24143.

[22] Loïc Paulevé & Adrien Richard (2011): *Static Analysis of Boolean Networks Based on Interaction Graphs: A Survey*. In: Proc. of SASB, Elsevier, pp. 93–104, doi:10.1016/J.ENTCS.2012.05.017.

[23] 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.

[24] Adrien Richard (2019): *Positive and negative cycles in Boolean networks*. J. Theor. Biol. 463, pp. 67–76, doi:10.1016/j.jtbi.2018.11.028.

[25] Adrien Richard & Elisa Tonello (2023): *Attractor separation and signed cycles in asynchronous Boolean networks*. Theor. Comput. Sci. 947, p. 113706, doi:10.1016/J.TCS.2023.113706.

[26] Julian D Schwab, Silke D Kühlwein, Nensi Ikonomi, Michael Kühl & Hans A Kestler (2020): *Concepts in Boolean network modeling: What do they all mean?* Comput. Struct. Biotechnol. J. 18, pp. 571–582, doi:10.1016/j.csbj.2020.03.001.

[27] René Thomas & Richard d'Ari (1990): *Biological feedback*. CRC press.

[28] Van-Giang Trinh & Belaid Benhamou (2024): *Static Analysis of Logic Programs via Boolean Networks*. arXiv:2407.09015.

[29] Van-Giang Trinh, Belaid Benhamou & Loïc Paulevé (2024): *mpbn: a simple tool for efficient edition and analysis of elementary properties of Boolean networks*. CoRR abs/2403.06255, doi:10.48550/ARXIV.2403.06255.

[30] 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.

[31] Van-Giang Trinh, Belaid Benhamou, Sylvain Soliman & François Fages (2024): *On the trap space semantics of logic programs, basic properties and relevance to program analysis*. Submitted paper.

[32] 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.

[33] 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.

Efficient OWL2QL Meta-reasoning Using ASP-based Hybrid Knowledge Bases

Haya Majid Qureshi Wolfgang Faber
University of Klagenfurt
Austria
{haya.qureshi,wolfgang.faber}@aau.at

Metamodeling refers to scenarios in ontologies in which classes and roles can be members of classes or occur in roles. This is a desirable modelling feature in several applications, but allowing it without restrictions is problematic for several reasons, mainly because it causes undecidability. Therefore, practical languages either forbid metamodeling explicitly or treat occurrences of classes as instances to be semantically different from other occurrences, thereby not allowing metamodeling semantically. Several extensions have been proposed to provide metamodeling to some extent. Building on earlier work that reduces metamodeling query answering to Datalog query answering, recently reductions to query answering over hybrid knowledge bases were proposed with the aim of using the Datalog transformation only where necessary. Preliminary work showed that the approach works, but the hoped-for performance improvements were not observed yet. In this work we expand on this body of work by improving the theoretical basis of the reductions and by using alternative tools that show competitive performance.

1 Introduction

Metamodeling helps in specifying conceptual modelling requirements with the notion of meta-classes (for instance, classes that are instances of other classes) and meta-properties (relations between meta-concepts). These notions can be expressed in OWL Full. However, OWL Full is so expressive for metamodeling that it leads to undecidability [13]. OWL 2 DL and its sub-profiles guarantee decidability, but they provide a very restricted form of metamodeling [7] and give no semantic support due to the prevalent Direct Semantics (DS).

Consider an example adapted from [6], concerning the modeling of biological species, stating that all golden eagles are eagles, all eagles are birds, and Harry is an instance of GoldenEagle, which further can be inferred as an instance of Eagle and Bird. However, in the species domain one can not just express properties of and relationships among species, but also express properties of the species themselves. For example "GoldenEagle is listed in the IUCN Red List of endangered species" states that GoldenEagle as a whole class is an endangered species. Note that this is also not a subclass relation, as Harry is not an endangered species. To formally model this expression, we can declare GoldenEagle to be an instance of new class EndangeredSpecies.

$$\mathit{Eagle} \sqsubseteq \mathit{Bird}, \quad \mathit{GoldenEagle} \sqsubseteq \mathit{Eagle}, \quad \mathit{GoldenEagle(Harry)}$$
$$\mathit{EndangeredSpecies} \sqsubseteq \mathit{Species}, \mathit{EndangeredSpecies(GoldenEagle)}$$

Note that the two occurrences of the IRI for GoldenEagle (in a class position and in an individual position) are treated as different objects in the standard direct semantics DS[1], therefore not giving semantic

[1] http://www.w3.org/TR/2004/REC-owl-semantics-20040210/

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 188–200, doi:10.4204/EPTCS.416.17

support to punned[2] entities and treating them as independent of each other by reasoners. These restrictions significantly limit meta-querying as well, since the underlying semantics for SPARQL queries over OWL 2 QL is defined by the *Direct Semantic Entailment Regime* [5], which uses *DS*.

To remedy the limitation of metamodeling, Higher-Order Semantics (HOS) was introduced in [10] for OWL 2 QL ontologies and later referred to as Meta-modeling Semantics (MS) in [11], which is the terminology that we will adopt in this paper. The interpretation structure of HOS follows the Hilog-style semantics of [1], which allows the elements in the domain to have polymorphic characteristics. Furthermore, to remedy the limitation of metaquerying, the Meta-modeling Semantics Entailment Regime (MSER) was proposed in [2], which does allow meta-modeling and meta-querying using SPARQL by reduction from query-answering over OWL 2 QL to Datalog queries.

In [15] several methods were proposed that reduce query-answering over OWL 2 QL to queries over hybrid knowledge bases instead. The idea there was to split the input ontology into two parts, one involving metamodeling and one that does not. The former is transformed to Datalog using the method of [2], while the latter is kept as an ontology and linked to the Datalog program. The precise bridge rules to be created were either all possible or just those relevant to the query (using an established module notion). Experiments using *HEXLite-owl-api-plugin* as a hybrid reasoner showed this to be a viable approach, even if the observed performance was not as quick as hoped for. This appeared to be due to internals of the hybrid reasoner and the lack of any query-oriented optimisations such as the magic set technique. Indeed, results in [14] indicate that absence of a query-oriented method is detrimental for performance.

In this work, we first recall the methods introduced in [15], then provide a detailed proof of correctness, and, most importantly, we use an extension of *DLV2* with *Python external atoms* as a hybrid reasoner. The system does support the magic set technique and our experiments show much better performance using this system.

2 Preliminaries

This section gives a brief overview of the language and the formalism used in this work.

2.1 OWL 2 QL

This section recalls the syntax of the ontology language OWL 2 QL and the *Metamodeling Semantics* (MS) for OWL 2 QL, as given in [12].

2.1.1 Syntax

We start by recalling some basic elements used for representing knowledge in ontologies: *Concepts*, a set of individuals with common properties, *Individuals*, objects of a domain of discourse, and *Roles*, a set of relations that link individuals. An OWL 2 ontology is a set of axioms that describes the domain of interest. The elements are classified into *literals* and *entities*, where *literals* are values belonging to datatypes and *entities* are the basic ontology elements denoted by *Internationalized Resource Identifiers* (IRI). The notion of the vocabulary V of an OWL 2 QL, constituted by the tuple $V = (V_e, V_c, V_p, V_d, D, V_i, L_{QL})$. In V, V_e is the union of V_c, V_p, V_d, V_i and its elements are called atomic expressions; V_c, V_p, V_d, and V_i are sets of IRIs, denoting, respectively, classes, object properties, data properties, and individuals, L_{QL} denotes the

[2]http://www.w3.org/2007/OWL/wiki/Punning

set of literals - characterized as OWL 2 QL datatype maps denoted as DM_{QL} and D is the set of datatypes in OWL 2 QL (including rdfs:Literal). Given a vocabulary V of an ontology $\mathcal{O}$, we denote by Exp the set of well formed expressions over V. For the sake of simplicity we use Description Logic (DL) syntax for denoting expressions in OWL 2 QL. Complex expressions are built over V, for instance, if $e_1, e_2 \in V$ then $\exists e_1.e_2$ is a complex expression. An OWL 2 QL Knowledge Base $\mathcal{O}$ is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is the TBox (inclusion axioms) and $\mathcal{A}$ is the ABox (assertional axioms). Sometimes we also let $\mathcal{O}$ denote $\mathcal{T} \cup \mathcal{A}$ for simplicity. OWL 2 QL is a finite set of logical axioms. The axioms allowed in an OWL 2 QL ontology have one of the forms: inclusion axioms $e_1 \sqsubseteq e_2$, disjointness axioms $e_1 \sqsubseteq \neg e_2$, axioms asserting property i.e., reflexive property $ref(e)$ and irreflexive property $irref(e)$ and assertional axioms i.e., $c(a)$ class assertion, , $p(a,b)$ object property assertion, and $d(a,b)$ data property assertion. We employ the following naming schemes (possibly adding subscripts if necessary): c,p,d,t denote a class, object property, data property and datatype. The above axiom list is divided into TBox axioms (further divided into positive TBox axioms and negative TBox axioms) and ABox axioms. The positive TBox axioms consist of all the inclusion and reflexivity axioms, the negative TBox axioms consist of all the disjointness and irreflexivity axioms and ABox consist of all the assertional axioms. For simplicity, we omit OWL 2 QL axioms that can be expressed by appropriate combinations of the axioms specified in the above axiom list. Also, for simplicity we assume to deal with ontologies containing no data properties.

2.1.2 Meta-modeling Semantics

The Meta-modeling Semantics (MS) is based on the idea that every entity in V may simultaneously have more than one type, so it can be a class, or an individual, or data property, or an object property or a data type. To formalise this idea, the Meta-modeling Semantics has been defined for OWL 2 QL. In what follows, $\mathbf{P}(S)$ denotes the power set of S. The meta-modeling semantics for $\mathcal{O}$ over V is based on the notion of interpretation, constituted by a tuple $\mathcal{I} = \langle \Delta, \cdot^I, \cdot^C, \cdot^P, \cdot^D, \cdot^T, \cdot^{\mathcal{I}} \rangle$, where

- Δ is the union of the two non-empty disjoint sets: $\Delta = \Delta^o \cup \Delta^v$, where Δ^o is the object domain, and Δ^v is the value domain defined by DM_{QL};
- $\cdot^I : \Delta^o \to \{True, False\}$ is a total function for each object $o \in \Delta^o$, which indicates whether o is an individual; if $\cdot^C, \cdot^P, \cdot^D, \cdot^T$ are undefined for an o, then we require $o^I = True$, also in other cases, e.g., if o is in the range of $\cdot^C$;
- $\cdot^C : \Delta^o \to \mathbf{P}(\Delta^o)$ is partial and can assign the extension of a class;
- $\cdot^P : \Delta^o \to \mathbf{P}(\Delta^o \times \Delta^o)$ is partial and can assign the extension of an object property;
- $\cdot^D : \Delta^o \to \mathbf{P}(\Delta^o \times \Delta^v)$ is partial and can assign the extension of a data property;
- $\cdot^T : \Delta^o \to \mathbf{P}(\Delta^v)$ is partial and can assign the extension of a datatype;
- $\cdot^{\mathcal{I}}$ is a function that maps every expression in Exp to Δ^o and every literal to Δ_v.

This allows for a single object o to be simultaneously interpreted as an individual via $\cdot^I$, a class via $\cdot^C$, an object property via $\cdot^P$, a data property via $\cdot^D$, and a data type via $\cdot^T$. For instance, for Example 1, $\cdot^C$, $\cdot^I$ would be defined for *GoldenEagle*, while $\cdot^P$, $\cdot^D$ and $\cdot^T$ would be undefined for it.

The semantics of logical axiom α is defined in accordance with the notion of axiom satisfaction for an MS interpretation $\mathcal{I}$. The complete set of notions is specified in Table 3.B in [12]. Moreover, $\mathcal{I}$ is said to be a model of an ontology $\mathcal{O}$ if it satisfies all axioms of $\mathcal{O}$. Finally, an axiom α is said to be logically implied by $\mathcal{O}$, denoted as $\mathcal{O} \models \alpha$, if it is satisfied by every model of $\mathcal{O}$.

2.2 Hybrid Knowledge Bases

Hybrid Knowledge Bases (HKBs) have been proposed for coupling logic programming (LP) and Description Logic (DL) reasoning on a clear semantic basis. Our approach uses HKBs of the form $\mathscr{K} = \langle \mathscr{O}, \mathscr{P} \rangle$, where $\mathscr{O}$ is an OWL 2 QL knowledge base and $\mathscr{P}$ is a hex program, as defined next.

Hex programs [3] extend answer set programs with external computation sources. We use hex programs with unidirectional external atoms, which import elements from the ontology of an HKB. For a detailed discussion and the semantics of external atoms, we refer to [4]. What we describe here is a simplification of the much more general hex formalism.

Regular atoms are of the form $p(X_1,\dots,X_n)$ where p is a predicate symbol of arity n and $X_1,\dots,X_n$ are terms, that is, constants or variables. An external atom is of the form $\&g[X_1,\dots,X_n](Y_1,\dots,Y_m)$ where g is an external predicate name g (which in our case interfaces with the ontology), $X_1,\dots,X_n$ are input terms and $Y_1,\dots,Y_m$ are output terms.

Next, we define the notion of positive rules that may contain external atoms.

Definition 1. *A hex rule r is of the form*

$$a \leftarrow b_1,\dots,b_k. \quad k \geq 0$$

where a is regular atom and $b_1,\dots,b_k$ are regular or external atoms. We refer to a as the head of r, denoted as $H(r)$, while the conjunction $b_1,...,b_k$ is called the body of r.

We call r ordinary if it does not contain external atoms. A program $\mathscr{P}$ containing only ordinary rules is called a positive program, otherwise a hex program. A hex program is a finite set of rules.

The semantics of hex programs generalizes the answer set semantics. The Herbrand base of $\mathscr{P}$, denoted $HB_{\mathscr{P}}$, is the set of all possible ground versions of atoms and external atoms occurring in $\mathscr{P}$ (obtained by replacing variables with constants). Note that constants are not just those in the standard Herbrand universe (those occuring in $\mathscr{P}$) but also those created by external atoms, which in our case will be IRIs from $\mathscr{O}$. Let the grounding of a rule r be $grd(r)$ and the grounding of program $\mathscr{P}$ be $grd(\mathscr{P}) = \bigcup_{r\in\mathscr{P}} grd(r)$. An interpretation relative to $\mathscr{P}$ is any subset $I \subseteq HB_{\mathscr{P}}$ containing only regular atoms. We write $I \models a$ iff $a \in I$. With every external predicate name $\&g \in G$ we associate an $(n+m+1)$-ary Boolean function $f_{\&g}$ (called oracle function) assigning each tuple $(I,x_1,\dots,x_n,y_1\dots,y_m)$ either 0 or 1, where I is an interpretation and x_i, y_j are constants. We say that $I \models \&g[x_1,\dots,x_n](y_1,\dots,y_m)$ iff $f_{\&g}(I,x_1\dots,x_n,y_1,\dots,y_m) = 1$. For a ground rule r, $I \models B(r)$ iff $I \models a$ for all $a \in B(r)$ and $I \models r$ iff $I \models H(r)$ whenever $I \models B(r)$. We say that I is a model of $\mathscr{P}$, denoted $I \models \mathscr{P}$, iff $I \models r$ for all $r \in grd(\mathscr{P})$. The *FLP-reduct* of $\mathscr{P}$ w.r.t I, denoted as $f\mathscr{P}^I$, is the set of all $r \in grd(\mathscr{P})$ such that $I \models B(r)$. An interpretation I is an answer set of $\mathscr{P}$ iff I is a minimal model of $f\mathscr{P}^I$. By $AS(\mathscr{P})$ we denote the set of all answer sets of $\mathscr{P}$. If $\mathscr{K} = \langle \mathscr{O}, \mathscr{P} \rangle$, then we write $AS(\mathscr{K}) = AS(\mathscr{P})$ — note that $\mathscr{O}$ is implicitly involved via the external atoms in $\mathscr{P}$. In this paper, $AS(\mathscr{K})$ will always contain exactly one answer set, so we will abuse notation and write $AS(\mathscr{K})$ to denote this unique answer set.

We will also need the notion of query answers of HKBs that contain rules defining a dedicated query predicate q. Given a hybrid knowledge base $\mathscr{K}$ and a query predicate q, let $ANS(q,\mathscr{K})$ denote the set $\{\langle x_1,\dots,x_n\rangle \mid q(x_1,\dots,x_n) \in AS(\mathscr{K})\}$.

3 Query Answering Using MSER

We consider SPARQL queries, a W3C standard for querying ontologies. While SPARQL query results can in general either be result sets or RDF graphs, we have restricted ourselves to simple **SELECT** queries, so it is sufficient for our purposes to denote results by set of tuples. For example, consider the following SPARQL query:

SELECT $?x\,?y\,?z$ WHERE {
  $?x\; rdf\!:\!type\; ?y.$
  $?y\; rdfs\!:\!SubClassOf\; ?z$
}

This query will retrieve all triples $\langle x,y,z\rangle$, where x is a member of class y that is a subclass of z. In general, there will be several variables and there can be multiple matches, so the answers will be sets of tuples of IRIs.

Now, we recall query answering under the Meta-modeling Semantics Entailment Regime (MSER) from [2]. This technique reduces SPARQL query answering over OWL 2 QL ontologies to Datalog query answering. The main idea of this approach is to define (i) a translation function τ mapping OWL 2 QL axioms to Datalog facts and (ii) a fixed Datalog rule base $\mathscr{R}^{ql}$ that captures inferences in OWL 2 QL reasoning.

The reduction employs a number of predicates, which are used to encode the basic axioms available in OWL 2 QL. This includes both axioms that are explicitly represented in the ontology (added to the Datalog program as facts via τ) and axioms that logically follow. In a sense, this representation is closer to a meta-programming representation than other Datalog embeddings that translate each axiom to a rule.

The function τ transforms an OWL 2 QL assertion α to a fact. For a given ontology $\mathscr{O}$, we will denote the set of facts obtained by applying τ to all of its axioms as $\tau(\mathscr{O})$; it will be composed of two portions $\tau(\mathscr{T})$ and $\tau(\mathscr{A})$, as indicated in Table 1.[3]

Table 1: τ Function

$\tau(\mathscr{O})$	α	$\tau(\alpha)$	α	$\tau(\alpha)$
$\tau(\mathscr{T})$	c1 $\sqsubseteq$ c2	isacCC(c1, c2)	r1 $\sqsubseteq \neg$ r2	disjrRR(r1,r2)
	c1 $\sqsubseteq \exists$r2$^-$.c2	isacCI(c1,r2,c2)	c1 $\sqsubseteq \neg$ c2	disjcCC(c1,c2)
	$\exists$r1 $\sqsubseteq \exists$r2.c2	isacRR(r1,r2,c2)	c1 $\sqsubseteq \neg\exists$r2$^-$	disjcCI(c1,r2)
	$\exists$r1$^-$ $\sqsubseteq$ c2	isacIC(r1,c2)	$\exists$r1$\sqsubseteq \neg$ c2	disjcRC(r1,c2)
	$\exists$r1$^-$ $\sqsubseteq \exists$r2.c2	isacIR(r1,r2,c2)	$\exists$r$_1$ $\sqsubseteq \neg\exists$r2	disjcRR(r1,r2)
	$\exists$r1$^-$ $\sqsubseteq \exists$r2$^-$.c2	isacII(r1,r2,c2)	$\exists$r1 $\sqsubseteq \neg\exists$r2$^-$	disjcRI(r1,r2)
	r1 $\sqsubseteq$ r2	isarRR(r1,r2)	$\exists$r1$^-$ $\sqsubseteq \neg$ c2	disjcIC(r1,c2)
	r1 $\sqsubseteq$ r2$^-$	isarRI(r1,r2)	$\exists$r1$^-$ $\sqsubseteq \neg\exists$r2	disjcIR(r1,r2)
	c1 $\sqsubseteq \exists$r2.c2	isacCR(c1,r2,c2)	$\exists$r1$^-$ $\sqsubseteq \neg\exists$r2$^-$	disjcII(r1,r2)
	$\exists$r1$\sqsubseteq$ c2	isacRC(r1,c2)	r1 $\sqsubseteq \neg$ r2$^-$	disjrRI(r1,r2)
	$\exists$r1 $\sqsubseteq \exists$r2$^-$.c2	isacRI(r1,r2,c2)	irref(r)	irrefl(r)
	refl(r)	refl(r)		
$\tau(\mathscr{A})$	c(x)	instc(c,x)	x $\neq$ y	diff(x,y)
	r(x, y)	instr(r,x,y)		

The fixed program $\mathscr{R}^{ql}$ can be viewed as an encoding of axiom saturation in OWL 2 QL. The full set of rules provided by authors of [2] are reported in the online repository of [14]. We will consider one rule to illustrate the underlying ideas:

[3]Note that there are no variables in $\tau(\mathscr{T})$ and $\tau(\mathscr{A})$.

$$\text{isacCR(C1,R2,C2)} \leftarrow \text{isacCC(C1,C3), isacCR(C3,R2,C2).}$$

The above rule encodes the following inference rule:

$$\mathcal{O} \models \text{C1} \sqsubseteq \text{C3}, \mathcal{O} \models \text{C3} \sqsubseteq \exists\text{R2.C2} \Rightarrow \mathcal{O} \models \text{C1} \sqsubseteq \exists\text{R2.C2}$$

Finally, the translation can be extended in order to transform conjunctive SPARQL queries under MS over OWL 2 QL ontologies into a Datalog query. SPARQL queries will be translated to Datalog rules using a transformation τ^q. τ^q uses τ to translate the triples inside the body of the SPARQL query $\mathcal{Q}$ and adds a fresh Datalog predicate q in the head to account for projections. In the following we assume q to be the query predicate created in this way.

For example, the translation of the SPARQL query given earlier will be

$$\text{q(X,Y,Z)} \leftarrow \text{instc(X,Y), isacCC(Y,Z).}$$

Given an OWL 2 QL ontology $\mathcal{O}$ and a SPARQL query $\mathcal{Q}$, let $ANS(\mathcal{Q},\mathcal{O})$ denote the answers to $\mathcal{Q}$ over $\mathcal{O}$ under MSER, that is, a set of tuples of IRIs. In the example above, the answers will be a set of triples.

4 MSER Query Answering via Hybrid Knowledge Bases

We propose four variants for answering MSER queries by means of Hybrid Knowledge Bases. We first describe the general approach and then define each of the four variants.

4.1 General Architecture

The general architecture is outlined in Figure 1. In all cases, the inputs are an OWL 2 QL ontology $\mathcal{O}$ and a SPARQL query $\mathcal{Q}$. We then differentiate between **OntologyFunctions** and **QueryFunctions**. The **OntologyFunctions** achieves two basic tasks: first, the ontology is split into two partitions $\mathcal{O}'$ and $\mathcal{O}''$, then $\tau(\mathcal{O}'')$ is produced.

The **QueryFunctions** work mainly on the query. First, a set $\mathcal{N}$ of IRIs is determined for creating *Interface Rules* (IR, simple hex rules), denoted as $\pi(\mathcal{N})$ for importing the extensions of relevant classes and properties from $\mathcal{O}'$. In the simplest case, $\mathcal{N}$, consist of all IRIs in $\mathcal{O}'$, but we also consider isolating those IRIs that are relevant to the query by means of Logic-based Module Extraction (LME) as defined in [8]. Then, τ^q translates $\mathcal{Q}$ into a Datalog query $\tau^q(\mathcal{Q})$. Finally, the created hex program components are united (plus the fixed inference rules), yielding the rule part $\mathcal{P} = \mathcal{R}^{ql} \cup \pi(\mathcal{N}) \cup \tau(\mathcal{O}'') \cup \tau^q(\mathcal{Q})$, which together with $\mathcal{O}'$ forms the HKB $\mathcal{K} = \langle \mathcal{O}', \mathcal{P} \rangle$, for which we then determine $ANS(q,\mathcal{K})$, where q is the query predicate introduced by $\tau^q(\mathcal{Q})$.

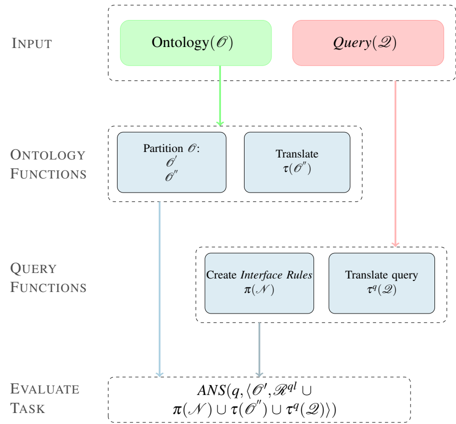
4.2 Basic Notions

Before defining the specific variations of our approach, we first define some auxiliary notions. The first definition identifies meta-elements.

Definition 2. *Given an Ontology $\mathcal{O}$, IRIs in $(V_c \cup V_p) \cap V_i$ are meta-elements, i.e., IRIs that occur both as individuals and classes or object properties.*

In our example, *GoldenEagle* is a meta-element. Meta-elements form the basis of our main notion, clashing axioms.

Figure 1: The Overall Architecture of Hybrid-Framework



Definition 3. *Clashing Axioms in $\mathscr{O}$ are axioms that contain meta-elements, denoted as* $\mathrm{CA}(\mathscr{O})$*. To denote clashing and non-clashing parts in TBox ($\mathscr{T}$) and ABox ($\mathscr{A}$), we write $\mathscr{A}^N = \mathscr{A} \setminus \mathrm{CA}(\mathscr{O})$ as non-clashing ABox, $\mathscr{A}^C = \mathrm{CA}(\mathscr{O}) \cap \mathscr{A}$ as clashing ABox; and likewise $\mathscr{T}^N = \mathscr{T} \setminus \mathrm{CA}(\mathscr{O})$ as non-clashing TBox and $\mathscr{T}^C = \mathrm{CA}(\mathscr{O}) \cap \mathscr{T}$ as clashing TBox.*

The clashing axiom notion allows for splitting $\mathscr{O}$ into two parts and generate $\mathscr{O}'$ without clashing axioms.

We would also like to distinguish between standard queries and meta-queries. A meta-query is an expression consisting of meta-predicates p and meta-variables v, where p can have other predicates as their arguments and v can appear in predicate positions. The simplest form of meta-query is an expression where variables appear in class or property positions also known as *second-order queries*. More interesting forms of meta-queries allow one to extract complex patterns from the ontology, by allowing variables to appear simultaneously in individual object and class or property positions. We will refer to non-meta-queries as standard queries. Moving towards *Interface Rules*, we first define signatures of queries, ontologies, and axioms.

Definition 4. *A* signature $\mathbf{S}(\mathscr{Q})$ *of a SPARQL query $\mathscr{Q}$ is the set of* IRIs *occurring in $\mathscr{Q}$. If no IRIs occur in $\mathscr{Q}$, we define $\mathbf{S}(\mathscr{Q})$ to be the signature of $\mathscr{O}$. Let $\mathbf{S}(\mathscr{O})$ (or $\mathbf{S}(\alpha)$) be the set of atomic classes, atomic roles and individuals that occur in $\mathscr{O}$ (or in axiom α).*

As hinted earlier, we can use $\mathbf{S}(\mathscr{O}')$ for creating interface rules ($\mathscr{O}'$ being the ontology part in the HKB), or use $\mathbf{S}(\mathscr{Q})$ for module extraction via *LME* as defined in [8] for singling out the identifiers relevant to the query, to be imported from the ontology via interface rules. We will denote this signature as $\mathbf{S}(LME(\mathbf{S}(\mathscr{Q}),\mathscr{O}'))$.

We next define the *Interface Rules* for a set of IRIs $\mathscr{N}$.

Definition 5. *For a set a of IRIs $\mathscr{N}$, let $\pi(\mathscr{N})$ denote the hex program containing a rule*

$$instc(C,X) \leftarrow \&g[C](X).$$

for each class identifier $C \in \mathscr{N}$, and a rule

$$instr(R,X,Y) \leftarrow \&g[R](X,Y).$$

for each property identifier $R \in \mathscr{N}$. Here &g is a shorthand for the external atom that imports the extension of classes or properties from the ontology $\mathscr{O}'$ of our framework.[4]

[4]Note that C and R above are not variables, but IRIs.

4.3 Variants

Now we define the four variants for the ontology functions, and two for the query functions. Since for one ontology function $\mathcal{O}'$ is empty, the two query functions have the same effect, and we therefore arrive at seven different variants for creating the hybrid knowledge bases (HKB).

The difference in the ontology functions is which axioms of $\mathcal{O} = \langle \mathcal{A}, \mathcal{T} \rangle$ stay in $\mathcal{O}'$ and which are in $\mathcal{O}''$, the latter of which is translated to Datalog. We use a simple naming scheme, indicating these two components:

$A\!-\!T$**:** $\mathcal{O}' = \mathcal{A}$, $\mathcal{O}'' = \mathcal{T}$.

$NAT\!-\!CAT$**:** $\mathcal{O}' = \langle \mathcal{A}^N, \mathcal{T} \rangle$, $\mathcal{O}'' = \langle \mathcal{A}^C, \mathcal{T} \rangle$.

$NAT\!-\!CACT$**:** $\mathcal{O}' = \langle \mathcal{A}^N, \mathcal{T} \rangle$, $\mathcal{O}'' = \langle \mathcal{A}^C, \mathcal{T}^C \rangle$.

$E\!-\!AT$**:** $\mathcal{O}' = \emptyset$, $\mathcal{O}'' = \mathcal{O} = \langle \mathcal{A}, \mathcal{T} \rangle$.

$E\!-\!AT$ serves as a baseline, as it boils down to the Datalog encoding of [2].

Definition 6. *Given $\mathcal{O} = \langle \mathcal{A}, \mathcal{T} \rangle$, let the $A\!-\!T$ HKB be $\mathcal{K}^{A-T}(\mathcal{O}) = \langle \mathcal{A}, \mathcal{R}^{ql} \cup \tau(\mathcal{T}) \rangle$; the $NAT\!-\!CAT$ HKB be $\mathcal{K}^{NAT-CAT}(\mathcal{O}) = \langle \langle \mathcal{A}^N, \mathcal{T} \rangle, \mathcal{R}^{ql} \cup \tau(\langle \mathcal{A}^C, \mathcal{T} \rangle) \rangle$; the $NAT\!-\!CACT$ HKB be $\mathcal{K}^{NAT-CACT}(\mathcal{O}) = \langle \langle \mathcal{A}^N, \mathcal{T} \rangle, \mathcal{R}^{ql} \cup \tau(\langle \mathcal{A}^C, \mathcal{T}^C \rangle) \rangle$; the $E\!-\!AT$ HKB be $\mathcal{K}^{E-AT}(\mathcal{O}) = \langle \emptyset, \mathcal{R}^{ql} \cup \tau(\mathcal{O}) \rangle$.*

Next we turn to the query functions. As hinted at earlier, we will consider two versions, which differ in the Interface Rules they create. Both create query rules $\tau^q(\mathcal{Q})$ for the given query, but one (*All*) will create interface rules for all classes and properties in the ontology part of the HKB, while the other (*Mod*) will extract the portion of the ontology relevant to query using *LME* and create Interface Rules only for classes and properties in this module.

For notation, we will overload the $\cup$ operator for HKBs, so we let $\langle \mathcal{O}, \mathcal{P} \rangle \cup \langle \mathcal{O}', \mathcal{P}' \rangle = \langle \mathcal{O} \cup \mathcal{O}', \mathcal{P} \cup \mathcal{P}' \rangle$ and we also let $\langle \mathcal{O}, \mathcal{P} \rangle \cup \mathcal{P}' = \langle \mathcal{O}, \mathcal{P} \cup \mathcal{P}' \rangle$ for ontologies $\mathcal{O}$ and $\mathcal{O}'$ and hex programs $\mathcal{P}$ and $\mathcal{P}'$.

Definition 7. *Given an HKB $\langle \mathcal{O}, \mathcal{P} \rangle$ and query $\mathcal{Q}$, let the All HKB be defined as $\mathcal{K}_{All}(\langle \mathcal{O}, \mathcal{P} \rangle, \mathcal{Q}) = \langle \mathcal{O}, \mathcal{P} \cup \tau^q(\mathcal{Q}) \cup \pi(\mathbf{S}(\mathcal{O})) \rangle$.*

Definition 8. *Given an HKB $\langle \mathcal{O}, \mathcal{P} \rangle$ and query $\mathcal{Q}$, let the Mod HKB be $\mathcal{K}_{Mod}(\langle \mathcal{O}, \mathcal{P} \rangle, \mathcal{Q}) = \langle \mathcal{O}, \mathcal{P} \cup \tau^q(\mathcal{Q}) \cup \pi(\mathbf{S}(LME(\mathbf{S}(\mathcal{Q}), \mathcal{O}))) \rangle$.*

We will combine ontology functions and query functions, and instead of $\mathcal{K}_\beta(\mathcal{K}^\alpha(\mathcal{O}), \mathcal{Q})$ we will write $\mathcal{K}^\alpha_\beta(\mathcal{O}, \mathcal{Q})$. We thus get eight combinations, but we will not use $\mathcal{K}^{E-AT}_{Mod}$, as it unnecessarily introduces Interface Rules. Also note that $\mathcal{K}^{E-AT}_{All}(\mathcal{O}, \mathcal{Q})$ does not contain any Interface Rules, because the ontology part of $\mathcal{K}^{E-AT}(\mathcal{O})$ is empty.

We will next show the correctness of the transformations. We start with the simplest case.

Proposition 1. *Let $\mathcal{O}$ be a consistent OWL 2 QL ontology and $\mathcal{Q}$ a conjunctive SPARQL query. Then, $ANS(\mathcal{Q}, \mathcal{O}) = ANS(q, \mathcal{K}^{E-AT}_{All}(\mathcal{O}, \mathcal{Q}))$, where q is the query predicate introduced by $\tau^q(\mathcal{Q})$.*

Proof. In [2] it was shown that $ANS(\mathcal{Q}, \mathcal{O}) = P^q(\tau(\mathcal{O})) = \{\langle x_1, \dots, x_n \rangle \mid q(x_1, \dots, x_n) \in MM(\mathcal{R}^{ql} \cup \tau(\mathcal{O}) \cup \tau^q(\mathcal{Q}))\}$.

Since $MM(P) = AS(P) = AS(\langle \emptyset, P \rangle)$ for any Datalog program P, it follows that $ANS(\mathcal{Q}, \mathcal{O}) = \{\langle x_1, \dots, x_n \rangle \mid q(x_1, \dots, x_n) \in AS(\langle \emptyset, \mathcal{R}^{ql} \cup \tau(\mathcal{O}) \cup \tau^q(\mathcal{Q}) \rangle)\}$.

Per definition, we get $\mathcal{K}^{E-AT}_{All}(\mathcal{O}, \mathcal{Q}) = \mathcal{K}_{All}(\mathcal{K}^{E-AT}(\mathcal{O}), \mathcal{O}, \mathcal{Q}) = \mathcal{K}_{All}(\langle \emptyset, \mathcal{R}^{ql} \cup \tau(\mathcal{O}) \rangle, \mathcal{O}, \mathcal{Q}) = \langle \emptyset, \mathcal{R}^{ql} \cup \tau(\mathcal{O}) \cup \tau^q(\mathcal{Q}) \cup \pi(\mathbf{S}(\mathcal{O})) \rangle$, therefore $ANS(q, \mathcal{K}^{E-AT}_{All}(\mathcal{O}, \mathcal{Q})) = \{\langle x_1, \dots, x_n \rangle \mid q(x_1, \dots, x_n) \in AS(\langle \emptyset, \mathcal{R}^{ql} \cup \tau(\mathcal{O}) \cup \tau^q(\mathcal{Q}) \cup \pi(\mathbf{S}(\mathcal{O})) \rangle)\}$.

We now show $AS(\langle\emptyset,\mathscr{R}^{ql}\cup\tau(\mathscr{O})\cup\tau^q(\mathscr{Q})\rangle) = AS(\langle\emptyset,\mathscr{R}^{ql}\cup\tau(\mathscr{O})\cup\tau^q(\mathscr{Q})\cup\pi(\mathbf{S}(\mathscr{O}))\rangle)$, which proves the proposition. Indeed, for any interpretation I we have that $I\not\models r$ for each $r\in\pi(\mathbf{S}(\mathscr{O}))$, because the ontology of the hybrid knowledge base is empty. Hence $f\langle\emptyset,\mathscr{R}^{ql}\cup\tau(\mathscr{O})\cup\tau^q(\mathscr{Q})\rangle^I = f\langle\emptyset,\mathscr{R}^{ql}\cup\tau(\mathscr{O})\cup\tau^q(\mathscr{Q})\cup\pi(\mathbf{S}(\mathscr{O}))\rangle^I$ for any interpretation I, and the equality of answer sets follows. □

Theorem 1. *Let $\mathscr{O}$ be a consistent OWL 2 QL ontology, $\mathscr{Q}$ a conjunctive SPARQL query, then it holds that $ANS(\mathscr{Q},\mathscr{O}) = ANS(q,\mathscr{K}^{\alpha}_{All}(\mathscr{O},\mathscr{Q}))$, where α is one of $A-T$, $NAT-CAT$, or $NAT-CACT$ and where q is the query predicate introduced by $\tau^q(\mathscr{Q})$.*

Proof. From Proposition 1 we have that $ANS(\mathscr{Q},\mathscr{O}) = ANS(q,\mathscr{K}^{E\text{–}AT}_{All}(\mathscr{O},\mathscr{Q}))$. We now show that $AS(\mathscr{K}^{E\text{–}AT}_{All}(\mathscr{O},\mathscr{Q})) = AS(\mathscr{K}^{\alpha}_{All}(\mathscr{O},\mathscr{Q}))$ and $ANS(q,\mathscr{K}^{E\text{–}AT}_{All}(\mathscr{O},\mathscr{Q})) = ANS(q,\mathscr{K}^{\alpha}_{All}(\mathscr{O},\mathscr{Q}))$ follows.

First, $\mathscr{K}^{E\text{–}AT}_{All}(\mathscr{O},\mathscr{Q}) = \langle\emptyset,\mathscr{R}^{ql}\cup\tau(\mathscr{O})\cup\tau^q(\mathscr{Q})\cup\pi(\mathbf{S}(\mathscr{O}))\rangle$ (for short E), and let $\mathscr{K}^{\alpha}_{All}(\mathscr{O},\mathscr{Q})) = \langle\mathscr{O}',\mathscr{R}^{ql}\cup\tau(\mathscr{O}'')\cup\tau^q(\mathscr{Q})\cup\pi(\mathbf{S}(\mathscr{O}))\rangle$ (for short A). In all cases, $\mathscr{O}'\subseteq\mathscr{O}$, $\mathscr{O}''\subseteq\mathscr{O}$ and $\mathscr{O}'\cup\mathscr{O}''=\mathscr{O}$. Moreover, $\mathscr{O}\models\varphi$ (φ atomic over $\mathbf{S}(\mathscr{O})$) if and only if $\mathscr{O}''\cup\{\psi \mid \mathscr{O}'\models\psi, \psi \text{ atomic over } \mathbf{S}(\mathscr{O})\}\models\varphi$, let us call this the ontology splitting property.

Now, for any interpretation I, $fE^I \neq fA^I$ may hold, but for any interpretation J, $J\models fE^I$ if and only if $J\models fA^I$. This is because for each atomic φ over $\mathbf{S}(\mathscr{O})$, either $\mathscr{O}'\models\varphi$, then there is a rule in $\pi(\mathbf{S}(\mathscr{O}))$ with a true body in fA^I (because of $\mathscr{O}'$) and $\tau(\varphi)$ in its head. That rule is satisfied by J iff $\tau(\varphi)\in J$. For fE^I, because of the results of [2] there is a rule in $\tau(\mathscr{O})$ with $\tau(\varphi)$ in its head and a true body; also that rule is satisfied by J iff $\tau(\varphi)\in J$. If $\mathscr{O}'\not\models\varphi$, then $\mathscr{O}''\cup\{\psi \mid \mathscr{O}'\models\psi, \psi \text{ atomic over } \mathbf{S}(\mathscr{O})\}\models\varphi$. In that case, the same rule with $\tau(\varphi)$ in its head is both in fA^I and fE^I.

Since $J\models fE^I$ if and only if $J\models fA^I$, also the minimal models of fE^I and fA^I are the same, and from this $AS(\mathscr{K}^{E\text{–}AT}_{All}(\mathscr{O},\mathscr{Q})) = AS(\mathscr{K}^{\alpha}_{All}(\mathscr{O},\mathscr{Q}))$ follows.

□

Note that the same proof also works for potential other variants that satisfy the ontology splitting property.

Theorem 2. *Let $\mathscr{O}$ be a consistent OWL 2 QL ontology, $\mathscr{Q}$ a conjunctive SPARQL query, then it holds that $ANS(\mathscr{Q},\mathscr{O}) = ANS(q,\mathscr{K}^{\alpha}_{Mod}(\mathscr{O},\mathscr{Q}))$, where α is one of $A-T$, $NAT-CAT$, or $NAT-CACT$ and where q is the query predicate introduced by $\tau^q(\mathscr{Q})$.*

Proof. Note that $LME(\mathbf{S}(\mathscr{Q}),\mathscr{O})$ is a module of $\mathscr{O}$ in the sense of [8]. This implies that for any atomic axiom φ over $\mathbf{S}(\mathscr{Q})$, $\mathscr{O}\models\varphi$ iff $LME(\mathbf{S}(\mathscr{Q}),\mathscr{O})\models\varphi$. It follows that $ANS(\mathscr{Q},\mathscr{O}) = ANS(\mathscr{Q},LME(\mathbf{S}(\mathscr{Q}),\mathscr{O})$. We have $ANS(\mathscr{Q},LME(\mathbf{S}(\mathscr{Q}),\mathscr{O}) = ANS(q,\mathscr{K}^{\alpha}_{All}(LME(\mathbf{S}(\mathscr{Q}),\mathscr{O}),\mathscr{Q}))$ from Theorem 1. $\mathscr{K}^{\alpha}_{Mod}(\mathscr{O},\mathscr{Q}) = \langle\mathscr{O}',\mathscr{R}^{ql}\cup\tau(\mathscr{O}'')\cup\tau^q(\mathscr{Q})\cup\pi(\mathbf{S}(LME(\mathbf{S}(\mathscr{Q}),\mathscr{O})))\rangle$ is very similar to $\mathscr{K}^{\alpha}_{All}(LME(\mathbf{S}(\mathscr{Q}),\mathscr{O}),\mathscr{Q}))$, which expands to $\langle LME(\mathbf{S}(\mathscr{Q}),\mathscr{O})',\mathscr{R}^{ql}\cup\tau(LME(\mathbf{S}(\mathscr{Q}),\mathscr{O})'')\cup\tau^q(\mathscr{Q})\cup\pi(\mathbf{S}(LME(\mathbf{S}(\mathscr{Q}),\mathscr{O})))\rangle$. $\mathscr{K}^{\alpha}_{Mod}(\mathscr{O},\mathscr{Q})$ just has the larger underlying ontology $\mathscr{O}$. $\mathscr{O}'$ may contain more axioms than $LME(\mathbf{S}(\mathscr{Q}),\mathscr{O})'$, but since the interface rules $\pi(\mathbf{S}(LME(\mathbf{S}(\mathscr{Q}),\mathscr{O})))$ are the same in both HKBs, they have no effect. Also $\tau(\mathscr{O}'')$ may contain more rules than $\tau(LME(\mathbf{S}(\mathscr{Q}),\mathscr{O})'')$, but none of them is relevant to q by definition. So eventually we get $ANS(q,\mathscr{K}^{\alpha}_{All}(LME(\mathbf{S}(\mathscr{Q}),\mathscr{O}),\mathscr{Q})) = ANS(q,\mathscr{K}^{\alpha}_{Mod}(\mathscr{O},\mathscr{Q})$, from which the result follows.

□

5 Evaluation

In [15] we conducted experiments using HEXLite with the OWL-API plugin. While it did show drastic improvements when using one of the hybrid approaches with respect to the baseline $E{-}AT$ and with using Mod rather then All, the absolute performance left to be desired. In particular, with the larger ontologies considered, no answer could be obtained even after hours. This contrasts sharply with the findings in [14], in which the best systems took only seconds to answer queries even on the larger ontologies. The main reasons appeared to be inefficiencies in the OWL-API plugin, paired with a lack of query-oriented computation.

In the meantime we became aware of *DLV2* with *Python external atoms*[5].

The version of DLV2 that we obtained from the developers directly supports the Turtle format of ontologies, and one can use ontology IRIs directly as predicate names. The rules in Definition 5 can then directly use class and role identifiers:

Definition 9. *For a set a of IRIs $\mathscr{N}$, let $\pi(\mathscr{N})$ denote the DLV2 program containing a rule*

$$instc(C,X) \leftarrow \; C(X).$$

for each class identifier $C \in \mathscr{N}$, and a rule

$$instr(R,X,Y) \leftarrow \; R(X,Y).$$

for each property identifier $R \in \mathscr{N}$.

For transforming our ontologies to Turtle format, we have used a utility called *ont-converter*[6] that automatically transforms the source ontology in different formats (RDF/XML, OWL/XML, N3, etc).

The experimental setting is the same as in [15]: we conducted two sets of experiments on the widely used Lehigh University Benchmark (LUBM) dataset and on the Making Open Data Effectively USable (MODEUS) Ontologies[7]. We only use the query function Mod, as it was evident in [15] that All has no advantage over Mod.

The **LUBM** datasets describe a university domain with information like departments, courses, students, and faculty. This dataset comes with 14 queries with different characteristics (low selectivity vs high selectivity, implicit relationships vs explicit relationships, small input vs large input, etc.). We have also considered the meta-queries *mq1*, *mq4*, *mq5*, and *mq10* from [9] as they contain variables in-property positions and are long conjunctive queries. We have also considered two special-case queries *sq1* and *sq2* from [2] to exercise the MSER features and identify the new challenges introduced by the additional expressivity over the ABox queries. Basically, in special-case queries, we check the impact of `DISJOINTWITH` and meta-classes in a query. For this, like in [2], we have introduced a new class named *TypeOfProfessor* and make *FullProfessor*, *AssociateProfessor* and *AssistantProfessor* instances of this new class and also define *FullProfessor*, *AssociateProfessor* and *AssistantProfessor* to be disjoint from each other. Then, in *sq1* we are asking for all those y and z, where y is a professor, z is a type of professor and y is an instance of z. In *sq2*, we have asked for different pairs of professors.

The **MODEUS** ontologies describe the *Italian Public Debt* domain with information like financial liability or financial assets to any given contracts [11]. It comes with 8 queries. These queries are pure meta-queries as they span over several levels of the knowledge base. MODEUS ontologies are meta-modeling ontologies with meta-classes and meta-properties.

[5] `https://dlv.demacs.unical.it/home`
[6] `https://github.com/sszuev/ont-converter`
[7] http://www.modeus.uniroma1.it/modeus/node/6

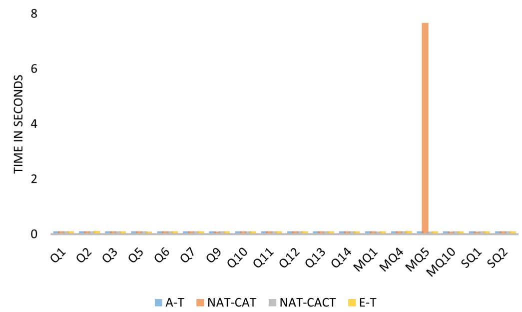


Figure 2: LUBM(1) experiments with standard and meta queries

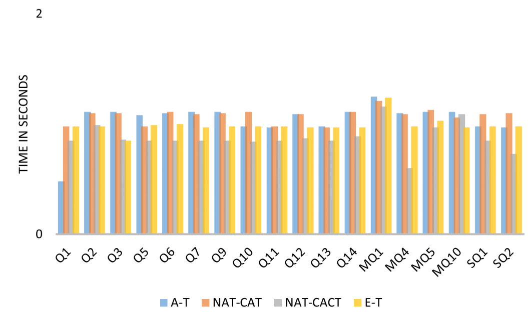


Figure 3: LUBM(9) experiments with Standard and Meta Queries

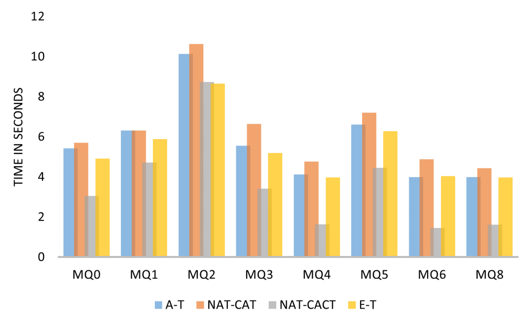


Figure 4: MODEUS(00) with Meta Queries

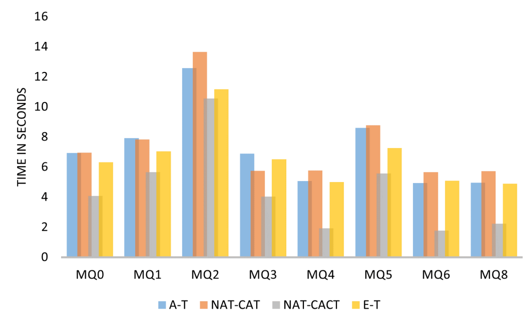


Figure 5: MODEUS(01) with Meta Queries

We have done the experiments on a Linux batch server, running Ubuntu 20.04.3 LTS (GNU/Linux 5.4.0-88-generic x86_64) on one AMD EPYC 7601 (32-Core CPU), 2.2GHz, Turbo max. 3.2GHz. The machine is equipped with 512GB RAM and a 4TB hard disk. Java applications used OpenJDK 11.0.11 with a maximum heap size of 25GB. During the course of the evaluation of the proposed variants we have used the time resource limitation as the benchmark setting on our data sets to examine the behavior of different variants. If not otherwise indicated, in both experiments, each benchmark had 3600 minutes (excluding the $\mathscr{K}$ generation time). For simplicity, we have not included queries that contain data properties in our experiments. We also have included the generation time of the hybrid knowledge base $\mathscr{K}$ including the loading of ontology and query, τ translation, module extraction, generating IR and translating queries. All material of experiments and results are available at `https://doi.org/10.5281/zenodo.13358935`.

In Figure 2 and 3, it can be seen that *DLV2* shows regular performance across all datasets and all variants of HKB with a slight increase in time depending on the size of the dataset. There is one outlier, meta-query MQ5 on LUBM(1) with $NAT-CAT$, which we were not expecting and might be a measurement error. In any case, this a massive improvement over the performance with HEXLite, where some of these queries required thousands of seconds to evaluate.

In Figures 4 to 7 the performance on MODEUS queries is reported. All the variants show consistent performance; however, the behaviour of the $NAT-CACT$ variant seems to be usually the best. These results are very satisfactory with respect to the results observed with HEXLite, where none of these queries were answered even after a few hours of runtime.

It should also be noted that $NAT-CACT$ with DLV2 also outperforms non-hybrid query answering

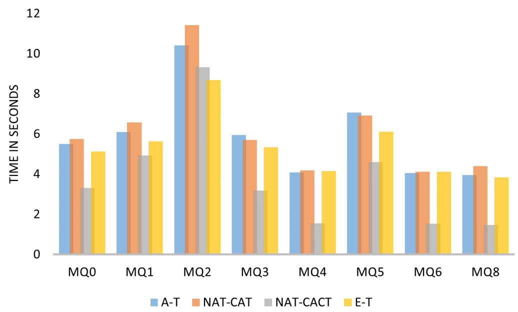


Figure 6: MODEUS(02) with Meta Queries

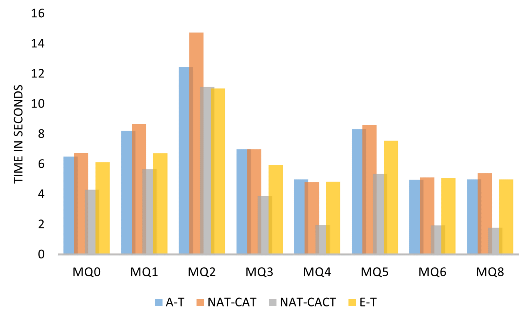


Figure 7: MODEUS(03) with Meta Queries

using DLV2 as reported in [14], making it the fastest known method on these ontologies and queries.

6 Discussion and Conclusion

This work shows that the methods introduced in [15] do not only have a positive relative impact when using a hybrid reasoner, but that they can also yield the best known performance when using a suitable tool for hybrid reasoning.

It seems clear from the result that there is a benefit of keeping some portions in the ontology rather than transforming the entire ontology to facts. This is, however, contingent of the availability of a query-aware method (in this case magic sets). Among the variants, $NAT-CACT$ showed best performance, which is also the one that hybridizes most.

In the future, we plan to investigate alternative variants for producing hybrid knowledge bases and assessing their performance. Another line of future work will be to identify more hybrid reasoning systems that are query aware and benchmark these.

References

[1] Weidong Chen, Michael Kifer & David S Warren (1993): *HiLog: A foundation for higher-order logic programming*. Journal of Logic Programming 15(3), pp. 187–230, doi:10.1016/0743-1066(93)90039-J.

[2] Gianluca Cima, Giuseppe De Giacomo, Maurizio Lenzerini & Antonella Poggi (2017): *On the SPARQL metamodeling semantics entailment regime for OWL 2 QL ontologies*. In: Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics, pp. 1–6, doi:10.1145/3102254.3102277.

[3] Thomas Eiter, Michael Fink, Giovambattista Ianni, Thomas Krennwallner, Christoph Redl & Peter Schüller (2016): *A model building framework for answer set programming with external computations*. Theory and Practice of Logic Programming 16(4), pp. 418–464, doi:10.1017/S1471068415000113.

[4] Thomas Eiter, Giovambattista Ianni, Roman Schindlauer & Hans Tompits (2006): *Effective integration of declarative rules with external evaluations for semantic-web reasoning*. In: European Semantic Web Conference, Springer, pp. 273–287, doi:10.1007/11762256_22.

[5] Birte Glimm (2011): *Using SPARQL with RDFS and OWL entailment*. In: Reasoning Web International Summer School, Springer, pp. 137–201, doi:10.1007/978-3-642-23032-5_3.

[6] Giancarlo Guizzardi, Joao Paulo Andrade Almeida, Nicola Guarino & Victorio Albani de Carvalho (2015): *Towards an Ontological Analysis of Powertypes*. In: JOWO@IJCAI, CEUR Workshop Proceedings 1517, CEUR-WS.org. Available at `https://ceur-ws.org/Vol-1517/JOWO-15_FOfAI_paper_7.pdf`.

[7] Pascal Hitzler, Markus Krötzsch & Sebastian Rudolph (2009): *Foundations of Semantic Web Technologies*. CRC press, doi:10.1201/9781420090512. Available at `http://www.semantic-web-book.org/`.

[8] Ernesto Jiménez-Ruiz, Bernardo Cuenca Grau, Ulrike Sattler, Thomas Schneider & Rafael Berlanga (2008): *Safe and economic re-use of ontologies: A logic-based methodology and tool support*. In: *European Semantic Web Conference*, Springer, pp. 185–199, doi:10.1007/978-3-540-68234-9_16.

[9] Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao & Michael Zakharyaschev (2014): *Answering SPARQL Queries over Databases under OWL 2 QL Entailment Regime*. In: *International Semantic Web Conference*, Springer, pp. 552–567, doi:10.1007/978-3-319-11964-9_35.

[10] Maurizio Lenzerini, Lorenzo Lepore & Antonella Poggi (2015): *A higher-order semantics for OWL 2 QL ontologies*. In: *Description Logics*, *CEUR Workshop Proceedings* 1350, CEUR-WS.org. Available at `https://ceur-ws.org/Vol-1350/paper-55.pdf`.

[11] Maurizio Lenzerini, Lorenzo Lepore & Antonella Poggi (2020): *Metaquerying made practical for OWL 2 QL ontologies*. *Information Systems* 88, p. 101294, doi:10.1016/J.IS.2018.02.012.

[12] Maurizio Lenzerini, Lorenzo Lepore & Antonella Poggi (2021): *Metamodeling and metaquerying in OWL 2 QL*. *Artificial Intelligence* 292, p. 103432, doi:10.1016/J.ARTINT.2020.103432.

[13] Boris Motik (2005): *On the properties of metamodeling in OWL*. In: *International Semantic Web Conference*, Springer, pp. 548–562, doi:10.1007/11574620_40.

[14] Haya Majid Qureshi & Wolfgang Faber (2021): *An Evaluation of Meta-reasoning over OWL 2 QL*. In: *RuleML+RR*, Springer, pp. 218–233, doi:10.1007/978-3-030-91167-6_15.

[15] Haya Majid Qureshi & Wolfgang Faber (2023): *Using Hybrid Knowledge Bases for Meta-reasoning over OWL 2 QL*. In: *Practical Aspects of Declarative Languages*, Springer, pp. 216–231, doi:10.1007/978-3-031-24841-2_14.

Pearce's Characterisation in an Epistemic Domain

Ezgi Iraz Su
Sinop University, Department of Computer Engineering, Sinop, Turkey
eirazsu@sinop.edu.tr

Answer-set programming (ASP) is a successful problem-solving approach in logic-based AI. In ASP, problems are represented as declarative logic programs, and solutions are identified through their answer sets. Equilibrium logic (EL) is a general-purpose nonmonotonic reasoning formalism, based on a monotonic logic called here-and-there logic. EL was basically proposed by Pearce as a foundational framework of ASP. Epistemic specifications (ES) are extensions of ASP-programs with subjective literals. These new modal constructs in the ASP-language make it possible to check whether a regular literal of ASP is true in every (or some) answer-set of a program. ES-programs are interpreted by world-views, which are essentially collections of answer-sets. (Reflexive) autoepistemic logic is a nonmonotonic formalism, modeling self-belief (knowledge) of ideally rational agents. A relatively new semantics for ES is based on a combination of EL and (reflexive) autoepistemic logic. In this paper, we first propose an overarching framework in the epistemic ASP domain. We then establish a correspondence between existing (reflexive) (auto)epistemic equilibrium logics and our easily-adaptable comprehensive framework, building on Pearce's characterisation of answer-sets as equilibrium models. We achieve this by extending Ferraris' work on answer sets for propositional theories to the epistemic case and reveal the relationship between some ES-semantic proposals.

1 Introduction

Answer-set programming (ASP), introduced by Gelfond&Lifschitz [7, 8], is an approach to declarative logic programming. Its reduct-based semantics is defined by *stable models* (alias, *answer-sets*), essentially the supported classical models of a logic program. ASP has demonstrated success in solving problems within logic-based AI: a problem is first encoded as a logic program, and then efficient ASP-solvers are employed to compute its stable models corresponding to the solutions. However, as Gelfond pointed out in his seminal work [4], ASP encounters challenges in accurately representing and reasoning about incomplete information. The difficulty arises when a program involves multiple stable models, and a proposition holds in one stable model but contradicts another. The main reason for this drawback lies in the local performance of ASP's negation as failure (NAF) operator, which handles incomplete information within individual stable models. To address this issue and enable collective reasoning about incomplete information, we need additional tools in the language of ASP. Epistemic modal operators provide one potential solution to ASP's limitation with incomplete information. By integrating such operators into the ASP-language, the new modal constructs in the extended language allow us to quantify over a collection of stable models and check whether a proposition holds in every (some) stable model.

The initial approach to this problem is by Gelfond's *epistemic specifications* [4, 5], referred to as ES_{94} here: Gelfond extended ASP with epistemic constructs known as *subjective literals*. Indeed, with the incorporation of epistemic modalities K and M, he could represent incomplete information within stable-model collections. While a subjective literal $\mathsf{K}\,l$ ($\mathsf{M}\,l$) makes it possible to check whether a literal l is true in every (some) stable model of a collection, in particular, the epistemic negation $\mathtt{not}\,\mathsf{K}$ accurately captures collective reasoning of incomplete information. The extended language is interpreted in terms of *world-views*, which are, in essence, stable-model collections. However, researchers have soon

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 201–214, doi:10.4204/EPTCS.416.18

realised that ES_{94} allows unsupported world-views. Thus, Gelfond himself [6], along with many other researchers, have proposed various semantic revisions for ES; each aiming to eliminate newly-appearing unintended results. The first counter-example that undermines the soundness of ES_{94}-semantics is the model $\{\{p\}\}$ resulting from the epistemic rule $p \leftarrow \mathsf{K}p$. This problem with recursion through K arises due to epistemic circular justification; yet efforts to resolve this problem do not focus on the core reasons for the emergence of unsupported models in ES_{94}. This situation leads to incrementally more complex reduct definitions. Although we refrain from calling these solutions ad hoc, as they can be based on reasonable grounds, we find it crucial to reveal the underlying reasons behind the existence of such models under ES_{94}-semantics. Moreover, we introduce a conventional and straightforward generalisation of ASP's reduct definition to epistemic logic programs, which constitutes our first contribution here.

One line of world-view computing methods in the literature depends on the reduct-based fixed-point techniques within the logic programming domain, with ES_{94} serving as the prototype and most subsequent formalisms being its follow-ups. In a parallel, purely logical context, world-views are computed as (reflexive) (auto)epistemic extensions of equilibrium models. The initial attempt in this direction was made by Wang&Zhang [19], whose semantics has captured the world-views of ES_{94}. Sequentially, stronger formalisms followed [18, 1, 17]. These epistemic equilibrium logics (EELs) share a common approach: a twofold world-view computation process. First, they determine stable models of an ES-program Π in terms of truth ($\mathtt{t}$) by applying the $\mathtt{t}$-minimality criterion of the formalism. This involves generalising the usual $\mathtt{t}$-minimality method which is used to compute stable models (equilibrium models) to ES-programs, resulting in the epistemic equilibrium models ($\mathtt{EEMs}$) of Π. The inclusion of epistemic constructs into the ASP-language requires the minimisation of these concepts as well, which is fundamental in nonmonotonic epistemic logics. Thus, once $\mathtt{t}$-stable models are determined, a knowledge-minimality technique should also be applied to guarantee stability in terms of knowledge ($\mathtt{k}$). As a result, world-views are stable-models w.r.t. both truth and knowledge. One formally strong $\mathtt{k}$-minimality approach applied to ES is Schwarz's [12] minimal model reasoning for nonmonotonic modal logics. Cabalar et al. [1] pioneered the introduction of this technique to ES, proposing a new semantics based on a combination of Pearce's equilibrium logic (EL) [11] and Schwarz's nonmonotonic KD45 [12, 16] (equivalently, Moore's autoepistemic logic). Their formalism so represents a nonmonotonic epistemic logic of belief where K is interpreted as *the self-belief of a rational agent*. It also captures ES_{94}-semantics under a foundedness restriction. Su [17] then suggested employing the reflexive closure of KD45-models, namely SW5-models, in the search for $\mathtt{k}$-minimal models and proposed reflexive autoepistemic EL. This formalism alternatively applies Schwarz's minimal model technique for nonmonotonic SW5 as a $\mathtt{k}$-minimality criterion, aligning it more closely with other ES-formalisms where the K operator formalises *knowledge*.

The existence of many ES-formalisms without a common agreement makes it difficult to understand the current state of the art. Thus, as a natural continuation, we explore the relationship between them. Our reference point will be classifying ES-formalisms under a twofold world-view computation method. We then generalise Ferraris' lemma, enabling *the capture of equilibrium models of a theory as its stable models*, to the epistemic case. Using our new result, we transform $\mathtt{EEMs}$ to truth-stable ($\mathtt{t}$-stable) models of epistemic ASP and vice versa. This work will then help ASP programmers better understand existing EELs, being reflected in the logic programming context and also give rise to a versatile and solid framework in epistemic ASP, an approach not studied before, which will be our main contribution here.

The rest of this paper is organised as follows: Sect. 2 provides preliminary information about ASP and Gelfond's primary ES_{94}-semantics. Sect. 3 presents epistemic ASP (EASP) as a unifying framework for several ES-semantics. Sect. 4 makes a short overview of the existing EELs in the literature, focusing on their $\mathtt{t}$-minimality methods. Sect. 5 establishes a correspondence between these EELs and EASP by generalising Ferraris' lemma to EASP. Sect. 6 concludes the paper with future work plan.

2 Background: ASP and epistemic specifications (ES) in a nutshell

In this context, ASP-formulas are built from an infinite set $\mathbb{P}$ of atoms using the connectives, viz. reversed implication ($\leftarrow$), disjunction ($\vee$), conjunction ($\wedge$), NAF ($\mathtt{not}$), strong negation ($\sim$), true ($\top$) and false ($\bot$). In ASP, a *literal* l is an atom p or a strongly-negated atom $\sim p$ for $p \in \mathbb{P}$. An ASP-program consists of a finite set of rules $\mathtt{r} : \mathtt{head}(\mathtt{r}) \leftarrow \mathtt{body}(\mathtt{r})$ s.t. $\mathtt{body}(\mathtt{r})$ is formed by a conjunction of literals possibly preceded by NAF, and $\mathtt{head}(\mathtt{r})$ is formed by a disjunction of literals: for $0 \le m \le n \le k$,

$$l_1 \vee \ldots \vee l_m \leftarrow l_{m+1} \wedge \ldots \wedge l_n \wedge \mathtt{not}\, l_{n+1} \wedge \ldots \wedge \mathtt{not}\, l_k . \tag{1}$$

Alternatively, we call body *goal* and its conjuncts *subgoals*. When $m = 0$, we suppose $\mathtt{head}(\mathtt{r})$ to be $\bot$ and call the rule $\mathtt{r}$ *constraint*. When $k = m$, we call $\mathtt{r}$ *fact* and omit both $\mathtt{body}(\mathtt{r})$ and $\leftarrow$. When $k = n$, we call $\mathtt{r}$ a positive rule. A program composed of only positive rules is positive. Finally, as strong negation can be removed from a logic program via auxiliary atoms, this paper mostly ignores $\sim$ for simplicity.

A *valuation* is a consistent (possibly empty) set T of literals, i.e., $p \notin T$ or $\sim p \notin T$ for any $p \in \mathbb{P}$. A valuation T satisfying an ASP-program Π (which means $T \models \Pi$) is a *classical* model of Π. Then, stable-models of Π are its reduct-based minimal classical models. Stable-model semantics is based on a program transformation that aims to eliminate '$\mathtt{not}$' from Π w.r.t. Π's classical model T (a candidate model), resulting in a positive program Π^T referred to as *reduct* of Π w.r.t. T: (*reduct-taking*) replace $\mathtt{not}\, p$ with $\top$ if $T \models \mathtt{not}\, p$ (equivalently, if $T \not\models p$, i.e., $p \notin T$); otherwise, with $\bot$. This approach also requires that the valuation T be a smallest (minimal) model of this reduct Π^T w.r.t. subset relation. Eventually, the successful models of this process are called *stable models* (alias, *answer-sets*) of Π.

2.1 Gelfond's epistemic specifications: ES_{94}

Epistemic specifications (ES) extends ASP-programs with the epistemic modal operators K ('known') and M ('may be true'). The language $\mathcal{L}_{\mathsf{ES}}$ contains four kinds of literals: *objective literals (l)*, *extended objective literals (L)*, *subjective literals (g)*, and *extended subjective literals (G)*, viz. for $p \in \mathbb{P}$,

l	L	g	G
$p \mid \sim p$	$l \mid \mathtt{not}\, l$	$\mathsf{K}l \mid \mathsf{M}l$	$g \mid \mathtt{not}\, g$

Note that ASP's regular literals are called objective literals in ES. By convention, the belief operator M can be defined in terms of the knowledge operator K, i.e., $\mathsf{M} \stackrel{\text{def}}{=} \mathtt{not}\,\mathsf{K}\,\mathtt{not}$, meaning that they are dual.

An ES-*rule* '$\mathtt{r} : l_1 \vee \ldots \vee l_m \leftarrow e_1 \wedge \ldots \wedge e_n$' is an extension of an ASP-rule (1) with extended subjective literals that can appear exclusively in $\mathtt{body}(\mathtt{r})$ as subgoals. Thus, $\mathtt{body}(\mathtt{r}) = e_1 \wedge \ldots \wedge e_n$ is a conjunction of arbitrary ES-literals. Then, an ES-*program* is a finite collection of ES-rules.

Truth conditions: Let $\mathcal{T}$ be a non-empty collection of valuations. Let I be a valuation, which is not necessarily included in $\mathcal{T}$. Then, for an objective literal l and a subjective literal g, we have:

$$\begin{array}{lll@{\qquad}lll}
\mathcal{T}, I \models l & \text{if} & l \in I; & \mathcal{T}, I \models \mathtt{not}\, l & \text{if} & l \notin I. \\
\mathcal{T}, I \models \mathsf{K}l & \text{if} & l \in T \text{ for every } T \in \mathcal{T}; & \mathcal{T}, I \models \mathtt{not}\, g & \text{if} & \mathcal{T}, I \not\models g. \\
\mathcal{T}, I \models \mathsf{M}l & \text{if} & l \in T \text{ for some } T \in \mathcal{T}; & & &
\end{array}$$

Note that the satisfaction of an objective literal l is independent of $\mathcal{T}$, while the satisfaction of a subjective literal g is independent of I. Thus, we simply write $\mathcal{T} \models g$ or $I \models l$. Then, we define the

Table 1: Kahl's reduct definition proposed in his PhD thesis [9] with changes over [6] in bold.

literal G	if $\mathcal{T} \models G$	if $\mathcal{T} \not\models G$	literal G	if $\mathcal{T} \models G$	if $\mathcal{T} \not\models G$
$\mathsf{K}\,l$	replace by l	replace by $\bot$	$\mathtt{not}\,\mathsf{K}\,l$	replace by $\top$	**replace by $\mathtt{not}\,l$**
$\mathsf{M}\,l$	replace by $\top$	**replace by $\mathtt{not}\,\mathtt{not}\,l$**	$\mathtt{not}\,\mathsf{M}\,l$	**replace by $\mathtt{not}\,l$**	replace by $\bot$

satisfaction of an ES-program Π as follows: $\mathcal{T}, I \models \Pi$ if for every rule $\mathtt{r} \in \Pi$, $\mathcal{T}, I \models \mathtt{r}$, i.e., explicitly

$$\mathcal{T}, I \models \mathtt{body}(\mathtt{r}) \quad \text{implies} \quad \mathcal{T}, I \models \mathtt{head}(\mathtt{r}).$$

An $\mathsf{S5}$-model is a nonempty collection of possible worlds, each with assigned truth values, where the connection between these worlds is defined by an equivalence relation (reflexive, symmetric, and transitive). In this context, we assume an $\mathsf{S5}$-model $\mathcal{T}$ to be in the form of a nonempty set of valuations s.t. any two valuations are related. When $\mathcal{T}, T \models \Pi$ for every $T \in \mathcal{T}$, we say that $\mathcal{T}$ is a *classical* $\mathsf{S5}$-*model* of Π. In particular, when we designate a valuation T s.t. $\mathcal{T}, T \models \Pi$, we call $(\mathcal{T}, T)$ a *pointed* $\mathsf{S5}$-model of Π. Extending this to a set $\mathcal{T}_0$ of designated valuations, $\langle \mathcal{T}, \mathcal{T}_0 \rangle$ is said to be a *multi-pointed* $\mathsf{S5}$-model of Π. To facilitate reading, we symbolise a multi-pointed $\mathsf{S5}$-model $\langle \mathcal{T}, \mathcal{T}_0 \rangle$ by underlying its designated valuations $T \in \mathcal{T}_0$ in an explicit representation of $\mathcal{T}$. Given $\mathcal{T} = \{\{a\}, \{b\}, \emptyset\}$, the (multi)pointed $\mathsf{S5}$-models $\langle \mathcal{T}, \{a\} \rangle$ and $\langle \mathcal{T}, \{\{a\}, \{b\}\} \rangle$ correspond to $\{\underline{\{a\}}, \{b\}, \emptyset\}$ and $\{\underline{\{a\}}, \underline{\{b\}}, \emptyset\}$ respectively. When no valuation is underlined or specified, by default this means that any valuation of $\mathcal{T}$ behaves as designated. The rest of the paper uses the terms "point", "valuation" and "world" interchangeably. Finally, given a syntactic ES-construct (head, rule, program, etc.) φ, when $\mathcal{T}, T \models \varphi$ for every $T \in \mathcal{T}$, we simply write $\mathcal{T} \models \varphi$.

Semantics: An ES-program Π is interpreted by means of its world-views, which are selected from among its $\mathsf{S5}$-models. Thus, given a candidate $\mathsf{S5}$-model $\mathcal{T}$ of Π, we first compute the (epistemic) reduct $\Pi^{\mathcal{T}} = \{\mathtt{r}^{\mathcal{T}} : \mathtt{r} \in \Pi\}$ of Π w.r.t. $\mathcal{T}$ by replacing every subjective literal $\mathsf{K}\,l$ ($\mathsf{M}\,l$), possibly preceded by $\mathtt{NAF}$, with $\top$ if $\mathcal{T} \models \mathsf{K}\,l (\mathsf{M}\,l)$; otherwise, with $\bot$. Then, $\mathcal{T}$ is a *world view* of Π if $\mathcal{T} = \mathtt{AS}(\Pi^{\mathcal{T}})$ where $\mathtt{AS}(\Pi)$ denotes the set of all stable models of Π. The reduct definition of ES_{94} is so oriented to remove extended subjective literals. The resulting program $\Pi^{\mathcal{T}}$ is then a nonepistemic, but not necessarily positive ASP-program potentially containing $\mathtt{NAF}$. In fact, ES_{94} offers a twofold reduct definition; first removing epistemic operators w.r.t. $\mathcal{T}$ and then eliminating $\mathtt{NAF}$ w.r.t. $T \in \mathcal{T} \cup \mathcal{X}$ akin to ASP's methodology.

2.1.1 Motivation

Example 1 The one-rule program $\Gamma = \{a \leftarrow \mathsf{K}\,a\}$ has 2 world-views, $\{\emptyset\}$ and $\{\{a\}\}$ in ES_{94}. Among these, only the former is intended. The self-supported model $\{\{a\}\}$ appears due to the fact that ES_{94}-reduct attacks positive (not preceded by $\mathtt{NAF}$) literals. This approach causes unsupported models to provide fake derivations for head-literals, which in return produce these models by fixed-point justifications. Thus, Gelfond's methodology includes flaws for programs containing cyclic dependencies like Γ, $\{a \leftarrow \mathsf{K}\,a \land \mathtt{not}\,\mathsf{K}\,b\}$, $\{a \leftarrow \mathsf{K}\,a \land \mathtt{not}\,b\}$, etc. Such circular scenarios may arise when the goal contains a positive subjective literal and is satisfied by the candidate unsupported $\mathsf{S5}$-model. Notice that transformation of a literal into true/false w.r.t. its truth-value is secure when it is preceded by $\mathtt{NAF}$ with literal reading *there is no evidence*, or when there exits logical derivations of literals as used by splitting property of (epistemic) ASP. To overcome this problem, Gelfond [6] slightly modifies his reduct definition by replacing $\mathsf{K}\,p$ with p when $\mathcal{T} \models \mathsf{K}\,p$ and partly avoids circular justifications, but the problem of recursion via M prevails.

This modification has probably necessitated further changes in his reduct definition as shown in Table1. The underlying reasons of Kahl's new reduct [9] may be grounded as follows: (1) If $\mathcal{T} \not\models \mathtt{not}\,\mathsf{K}\,l$, then $\mathcal{T} \models \mathsf{K}\,l$. When the reduct definition transforms $\mathsf{K}\,l$ into l, it replaces $\mathtt{not}\,\mathsf{K}\,l$ with $\mathtt{not}\,l$. (2) Remember that $\mathsf{M} \overset{\text{def}}{=} \mathtt{not}\,\mathsf{K}\mathtt{not}$. If $\mathcal{T} \not\models \mathsf{M}\,l$, then $\mathcal{T} \not\models \mathtt{not}\,\mathsf{K}\mathtt{not}\,l$, i.e., $\mathcal{T} \models \mathsf{K}\mathtt{not}\,l$. A similar reasoning may force the transformation of $\mathsf{K}\mathtt{not}\,l$ into $\mathtt{not}\,l$; $\mathsf{M}\,l$ into $\mathtt{not}\,\mathtt{not}\,l$. (3) If $\mathcal{T} \models \mathtt{not}\,\mathsf{M}\,l$, then $\mathcal{T} \models \mathsf{K}\mathtt{not}\,l$. If $\mathsf{K}\mathtt{not}\,l$ is transformed into $\mathtt{not}\,l$, then $\mathtt{not}\,\mathsf{M}\,l$ is turned into $\mathtt{not}\,\mathtt{not}\,\mathtt{not}\,l$, equivalently [10] into $\mathtt{not}\,l$. While this explanation is a guess, in fact when NAF is involved, such further intricate changes may not be required.

Example 2 Another recursive program $\Sigma = \{a \leftarrow \mathsf{M}\,a\}$ yields the same world-views in ES_{94}. Researchers have widely varying perspectives on the intended models of Σ. While some find both models reasonable, the others argue that Σ should have one model; yet they also differ on which model should be preferred. We will not engage in this debate, as different approaches may prove useful depending on the specific problem at hand. Our stance on the topic is distinct. In alignment with Su et al.'s approach [18], and following the tradition of intuitionistic modal logics, we will adopt a positive belief operator here, namely $\hat{\mathsf{K}}$, which is not definable in terms of K and $\mathtt{not}$. As $\mathsf{M} \overset{\text{def}}{=} \mathtt{not}\,\mathsf{K}\mathtt{not}$, in our opinion, M cannot be regarded as purely positive like $\mathtt{not}\,\mathtt{not}\,a$ in ASP. Remember that Su et al. handle M as a syntactic sugar, giving a concise representation for the equivalent formulas $\mathtt{not}\,\mathtt{not}\,\hat{\mathsf{K}}$, $\hat{\mathsf{K}}\mathtt{not}\,\mathtt{not}$, and $\mathtt{not}\,\mathsf{K}\mathtt{not}$. Also recall that in epistemic ASP, aligning with ASP, double NAF should not vanish regardless of where it occurs. On the other hand, similarly to Γ in Ex. 1, we claim that the intended model of $\Sigma' = \{a \leftarrow \hat{\mathsf{K}}\,a\}$ should be $\{\emptyset\}$.

3 Epistemic Answer Set Programming (EASP)

This section introduces a direct generalisation of logic programs under stable-model semantics (aka, ASP-programs) to epistemic logic programs under stable S5-model semantics. This new concept has been partially explored by [15]. The shift from the general term *world-view* to *stable* S5-*model* in EASP, and *equilibrium* S5-*model* in the following section is intended to emphasise the purpose of this work. Our main motivation for this study arises from the unsupported models that emerge due to circular justifications under ES_{94}-semantics (see Ex. 1-2). ES_{94}'s reduct definition deviates somewhat from the traditional approach. We here propose a new reduct definition for ES-programs, oriented to eliminate exclusively NAF. Thus, our reduct is a positive program, similar to the method in search for stable models.

The new approach exploits a two-step computation process, focusing on stability in terms of truth ($\mathtt{t}$) and knowledge ($\mathtt{k}$). The method involves finding the minimal models in terms of truth first, and then refining them further w.r.t. a $\mathtt{k}$-minimality criterion to select stable S5-models. Such models then capture truth and knowledge minimality concepts that is central in (nonmonotonic) epistemic ASP. In broader terms, what we refer to as $\mathtt{t}$-minimality in ES is essentially an extension of the familiar minimisation criterion of ASP from classical models to classical S5-models. However, $\mathtt{k}$-minimality is a relatively new concern within the ASP field compared to the well-established method of $\mathtt{t}$-minimality. The necessity for such a technique has become evident with the incorporation of epistemic concepts into ASP and the need to maximise epistemic possibilities (i.e., ignorance).

A stable S5-model $\mathcal{T}$ of an epistemic logic program Π is its S5-model s.t. each valuation $T \in \mathcal{T}$ forms Π's pointed S5-model $(\mathcal{T}, T)$ where T is minimal w.r.t. truth and $\mathcal{T}$ is minimal w.r.t. knowledge. For a nonepistemic ASP-program Π, such valuations are Π's stable-models in ASP, and the (unique) stable S5-model $\mathcal{T}$ is the set of all such models. Similar to stable-models of ASP, the intuition underlying stable S5-models is to capture the *rationality* of an agent associated with an epistemic logic program Π: "*an agent is not supposed to believe anything that it is not forced to believe.*" The aim, in principle, is to determine which propositions can be nonmonotonically inferred from Π by considering all its stable-models. These inferences are then used to deduce new information about the knowledge of Π.

3.1 The Language of EASP ($\mathcal{L}_{\mathsf{EASP}}$)

The language $\mathcal{L}_{\mathsf{EASP}}$ extends that of ASP by epistemic modalities K and $\hat{\mathsf{K}}$. Literals (λ) of $\mathcal{L}_{\mathsf{EASP}}$ are of two types; *objective (l)* and *subjective (g)* literals, viz. $l := p \mid {\sim}p$ and $g := \mathsf{K}l \mid \hat{\mathsf{K}}l$ for $p \in \mathbb{P}$. Then, $\mathtt{not}\,\lambda$ means *failing to derive* λ, *the query* λ? *is undetermined and assumed to be false*; yet we do not offer literal interpretations of the modalities for the sake of flexibility.

Replacing literals of ASP with those of EASP in (1), we obtain an EASP-*rule* $\mathtt{r}$, viz.

$$\lambda_1 \vee \ldots \vee \lambda_m \leftarrow \lambda_{m+1} \wedge \ldots \wedge \lambda_n \wedge \mathtt{not}\,\lambda_{n+1} \wedge \ldots \wedge \mathtt{not}\,\lambda_k. \quad (\text{for } 0 \le m \le n \le k) \tag{2}$$

in which λ_i's are objective or subjective literals for every $i = 1,\ldots,k$. When we restrict λ_i's to objective literals, the resulting program is a disjunctive logic program [8]. Hence, EASP-rules are conservative extensions of ASP's disjunctive rules (1). Different from ES, we allow $\mathsf{K}l$ and $\hat{\mathsf{K}}l$ to appear in $\mathtt{head}(\mathtt{r})$. While the use of subjective literals in the head has not yet been fully explored, we still find it useful to provide the same syntax structure with ASP for easier understanding of the approach. This way, extensions to richer languages are straightforward via the main ASP track. An *epistemic logic program* (ELP), also known as EASP-program, is a finite collection of EASP-rules (2).

3.2 Semantics of EASP in terms of stable S5-models

We first introduce $\mathtt{t}$-minimality concept in EASP. Based on the existing ES-formalisms in the literature, we provide two slightly different approach. For example, the program $\Phi = \{\mathtt{r}_1, \mathtt{r}_2, \mathtt{r}_3\}$

$$\mathtt{r}_1 = a \vee b. \qquad \mathtt{r}_2 = a \leftarrow \mathsf{K}b. \qquad \mathtt{r}_3 = b \leftarrow \mathsf{K}a. \tag{3}$$

may produce $\mathtt{t}$-minimal models $\mathcal{T}_1 = \{\{a\},\{b\}\}$ and $\mathcal{T}_2 = \{\{a,b\}\}$; yet it may also yield $\mathcal{T}_1$ only, depending on how restrictive we want to be. In EASP, this subtle distinction originates from differing approaches of $\mathtt{t}$-minimality techniques, emphasising functional vs. relational perspective.

Definition 1 (weakening of a point in an S5-model in terms of truth: functional approach) Given a nonempty collection $\mathcal{T}$ of valuations, let $\mathsf{s} : \mathcal{T} \to 2^{\mathbb{P}}$ be a *subset* function s.t. $\mathsf{s}(T) \subseteq T$ for every $T \in \mathcal{T}$. Let *id* refer to the identity function, and let $\mathsf{s}[\mathcal{T}] = \{\mathsf{s}(T)\}_{T\in\mathcal{T}}$ denote the image of $\mathcal{T}$ under s. A *functional ($\mathtt{f}$) weakening* of $\mathcal{T}$ at a point $T \in \mathcal{T}$ by means of s is identified with $\langle \mathsf{s}[\mathcal{T}], \mathsf{s}(T)\rangle$ s.t. $\mathsf{s} \neq id$ on $\mathcal{T}$ and $\mathsf{s}|_{\mathcal{T}\setminus\{T\}} = id$, by which we take a strict subset of $T \in \mathcal{T}$ and keep the elements of $\mathcal{T} \setminus \{T\}$ unchanged. We say that $\langle \mathsf{s}[\mathcal{T}], \mathsf{s}(T)\rangle$ is $\mathtt{f}$*-weaker* than $\langle \mathcal{T}, T\rangle$ on $T \in \mathcal{T}$ and denote it by $\langle \mathsf{s}[\mathcal{T}], \mathsf{s}(T)\rangle \lhd_{\mathtt{f}} \langle \mathcal{T}, T\rangle$.

Def. 1 has already been introduced by [15]; yet the following more cautious approach is novel.

Definition 2 (weakening of a point in an S5-model in terms of truth: relational approach) Let $\mathsf{s}_{\mathtt{r}} : \mathcal{T} \Rightarrow 2^{\mathbb{P}}$ be a multi-valued *subset* function s.t. $\mathsf{s}_{\mathtt{r}}(T) \subseteq 2^T$ and $\mathsf{s}_{\mathtt{r}}(T) \neq \emptyset$ for every $T \in \mathcal{T}$. For ease of understanding, we also design $\mathsf{s}_{\mathtt{r}}$ as a serial subset relation, relating each $T \in \mathcal{T}$ to at least one element from 2^T and form the collection $\mathsf{s}_{\mathtt{r}} = \{(T,H) : H \in \mathsf{s}_{\mathtt{r}}(T)\}_{T\in\mathcal{T}}$. Then, a *relational ($\mathtt{r}$) weakening* of $\mathcal{T}$ at a point $T \in \mathcal{T}$ by means of $\mathsf{s}_{\mathtt{r}}$ is identified with $\langle \mathsf{s}_{\mathtt{r}}[\mathcal{T}], \mathsf{s}_{\mathtt{r}}(T)\rangle$ s.t. $\mathsf{s} \neq id$ on $\mathcal{T}$ and $\mathsf{s}|_{\mathcal{T}\setminus\{T\}} = id$, by which we replace only T in $\mathcal{T}$ by a set of its subsets including at least one strict subset $H \subset T$. We say that $\langle \mathsf{s}_{\mathtt{r}}[\mathcal{T}], \mathsf{s}_{\mathtt{r}}(T)\rangle$ is $\mathtt{r}$*-weaker* than $\langle \mathcal{T}, T\rangle$ on $T \in \mathcal{T}$ and denote it by $\langle \mathsf{s}_{\mathtt{r}}[\mathcal{T}], \mathsf{s}_{\mathtt{r}}(T)\rangle \lhd_{\mathtt{r}} \langle \mathcal{T}, T\rangle$.

We now define a nonmonotonic satisfaction relation $\models^*$ for S5-models, involving a $\mathtt{t}$-minimality criterion based on set inclusion over each set $T \in \mathcal{T}$. Intuitively, this condition says that none of the weakenings of $\langle \mathcal{T}, T\rangle$ is an S5-model of an epistemic logic program (ELP) Π for every $T \in \mathcal{T}$.

Definition 3 (generalisation of the truth-minimality ($\mathtt{t}$-minimality) criterion of ASP to EASP) For a positive EASP-program Π, let $\mathcal{T}$ be a nonempty collection of valuations, and $T \in \mathcal{T}$. Then, we have:

$$\mathcal{T},T \models^*_{\mathtt{f}} \Pi \quad \text{iff} \quad \mathcal{T},T \models \Pi \text{ and } \mathtt{s}[\mathcal{T}],\mathtt{s}(T) \not\models \Pi \text{ for every } \mathtt{s} \text{ s.t. } \langle \mathtt{s}[\mathcal{T}],\mathtt{s}(T)\rangle \lhd_{\mathtt{f}} \langle \mathcal{T},T\rangle. \tag{4}$$

Thus, $\mathcal{T}$ is a $\mathtt{t}_{\mathtt{f}}$*-minimal* model of Π if $\mathcal{T},T \models^*_{\mathtt{f}} \Pi$ for every $T \in \mathcal{T}$ [15]. In this paper, we also define $\models^*_{\mathtt{r}}$ by replacing $\mathtt{s}$ with $\mathtt{s}_{\mathtt{r}}$, and $\lhd_{\mathtt{f}}$ with $\lhd_{\mathtt{r}}$ in (4) and produce $\mathtt{t}_{\mathtt{r}}$*-minimal* models of Π accordingly.

Although the above definitions seem to be technically complex and daunting, they are easily applied:

Example 3 Reconsider first the program Φ, identified by (3), and its S5-model $\mathcal{T}_2 = \{\{a,b\}\}$. Then construct $2^{\{a,b\}} = \{\{a,b\},\{a\},\{b\},\emptyset\}$. Since the $\mathtt{f}$-weaker models $\{\{a\}\}$, $\{\{b\}\}$, and $\{\emptyset\}$ of $\mathcal{T}_2$ do not satisfy $\mathtt{r}_3$, $\mathtt{r}_2$, and $\mathtt{r}_1$ respectively, Φ does not hold in them either. Thus, $\mathcal{T}_2$ is a $\mathtt{t}_{\mathtt{f}}$-minimal model of Φ.

What eliminates $\mathcal{T}_2$ in the second approach is the relational nature of the weakening methodology because now we have to consider all possible subsets of $2^{\{a,b\}}$ different from $\emptyset$ and $\mathcal{T}_2$, i.e., all the elements of the set $2^{2^{\{a,b\}}} \setminus \{\mathcal{T}_2,\emptyset\}$. The element $\{\{a\},\{b\}\}$ from this set, namely an $\mathtt{r}$-weakening of $\mathcal{T}_2$ at the point $\{a,b\} \in \mathcal{T}_2$, satisfies Φ. Thus, $\mathcal{T}_2$ fails to be a $\mathtt{t}_{\mathtt{r}}$-minimal model of Φ.

Note that when we consider $\mathcal{T}_1$, different from the singleton model $\mathcal{T}_2$, we follow the above steps for every pointed S5-model of $\mathcal{T}_1$, viz. $\{\underline{\{a\}},\{b\}\}$ and $\{\{a\},\underline{\{b\}}\}$. Also note that Φ is a positive program, and its reduct trivially equals itself. Thus, our reduct is not interested in the positive literals $\mathsf{K}a$ and $\mathsf{K}b$ in Φ.

Fact 1 Functional minimality implies relational minimality because any function can be defined as a relation. Thus, a $\mathtt{t}_{\mathtt{r}}$-minimal model of an ELP Π is a $\mathtt{t}_{\mathtt{f}}$-minimal model of Π, but not vice versa.

Example 4 Consider the EASP-program $\Sigma = \{\mathtt{r}_1,\mathtt{r}_2,\mathtt{r}_3,\mathtt{r}_4\}$ with its rules explicitly represented below:

$$\mathtt{r}_1 = a \vee b. \qquad \mathtt{r}_2 = c \leftarrow b. \qquad \mathtt{r}_3 = d \leftarrow \mathsf{K}a. \qquad \mathtt{r}_4 = \bot \leftarrow \hat{\mathsf{K}}d.$$

Note that Σ is a positive program. We compute that $\{\{a\},\{b,c\}\}$ is a $\mathtt{t}$-minimal model of Σ: $\{\underline{\{a\}},\{b,c\}\} \models \Sigma$ while its only $\mathtt{f}$-weakening $\{\underline{\emptyset},\{b,c\}\}$ refutes it. Likewise, $\{\{a\},\underline{\{b,c\}}\} \models \Sigma$ while all its $\mathtt{f}$-weakenings, i.e., $\{\{a\},\underline{\{b\}}\}$, $\{\{a\},\underline{\{c\}}\}$, and $\{\{a\},\underline{\emptyset}\}$ do not satisfy it. We leave it to the reader to show that $\mathcal{T}$ is also $\mathtt{t}_{\mathtt{r}}$-minimal; yet we give a hint that while computing the $\mathtt{r}$-weakenings of, for example, $\{\{a\},\underline{\{b,c\}}\}$, we consider all possible models including $\{\{a\},\underline{\{b\}},\underline{\{c\}}\}$, $\{\{a\},\underline{\{b,c\}},\underline{\emptyset},\underline{\{b\}}\}$, etc. There are 14 of such models. Clearly, $\{\{b,c\}\}$ is Σ's other $\mathtt{t}$-minimal model, that is unintended and to be eliminated under $\mathtt{k}$-minimality conditions. Note that like $\mathsf{K}a$, the other positive literal $\hat{\mathsf{K}}d$ is not involved in the reduct-taking process.

Remark 1 The need for relational minimality arises from the fact that under singleton S5-models like $\{\{p\}\}$, the literals $\mathsf{K}p$, $\hat{\mathsf{K}}p$, and p are of no difference since quantification is trivially performed over just one valuation $\{p\}$. For instance, notice that when we replace $\mathsf{K}l$ by l in Φ (3), the resulting ASP-program has the stable model $\{a,b\}$. Using relational weakening, we increase epistemic possibilities (points) while reducing truth. Quantifying over these points then reveals the nontrivial functionality of subjective literals. In a sense, the relational $\mathtt{t}$-minimality approach simultaneously embeds in itself a kind of $\mathtt{k}$-minimality strategy by increasing ignorance with epistemic possibilities. The difference between two minimality methods strikingly appears for Φ under the S5-model $\{\{a,b\}\}$ (see Ex. 3). Adding the constraint $\mathtt{r}_{\mathtt{c}} = \bot \leftarrow \mathtt{not}\,\mathsf{K}a$ into Φ, the new program $\Phi' = \Phi \cup \{\mathtt{r}_{\mathtt{c}}\}$ has a world-view $\{\{a,b\}\}$ under several ES-formalisms. Some researchers find this result unsupported; yet the existing $\mathtt{k}$-minimality techniques is unable to eliminate this model. Thus, a more restrictive $\mathtt{t}$-minimality tool has been designed to remove models like $\{\{a,b\}\}$ while computing $\mathtt{t}$-minimal models. We do not discuss this issue here, as our aim is just to establish a correspondence between existing ES-formalisms; to put it better, to demonstrate the reader how current epistemic equilibrium logics are manifested in the logic programming domain.

We will now see how to compute stable w.r.t. truth ($\mathtt{t}$-stable) models of an arbitrary EASP program potentially including $\mathtt{NAF}$. Satisfaction of the subjective literal $\hat{\mathsf{K}}l$ is the same as $\mathsf{M}l$ in ES. What makes the difference is primarily how the reduct definition handles them.

Definition 4 (generalisation of the conventional reduct definition of ASP to EASP) For an arbitrary EASP-program Π, let $\mathcal{T}$ be a nonempty collection of valuations, and let $T \in \mathcal{T}$. Then, the reduct $\Pi^{\langle \mathcal{T}, T \rangle}$ of Π w.r.t. the pointed $\mathsf{S5}$-model $\langle \mathcal{T}, T \rangle$ is defined by replacing every occurrence of $\mathtt{NAF}$-negated (i.e., preceded by $\mathtt{NAF}$) literals $\mathtt{not}\,\lambda$ in Π with the truth-constants

$$\begin{array}{lll} \bot \text{ if } \mathcal{T}, T \models \lambda & (\text{ for } \lambda = l \text{ if } T \models l; & \text{for } \lambda = \mathsf{K}l\,(\hat{\mathsf{K}}l) \text{ if } \mathcal{T} \models \mathsf{K}l\,(\hat{\mathsf{K}}l)\ ); \\ \top \text{ if } \mathcal{T}, T \not\models \lambda & (\text{ for } \lambda = l \text{ if } T \not\models l; & \text{for } \lambda = \mathsf{K}l\,(\hat{\mathsf{K}}l) \text{ if } \mathcal{T} \not\models \mathsf{K}l\,(\hat{\mathsf{K}}l)\ ). \end{array}$$

Thus, $\mathcal{T}$ is a $\mathtt{t}$-*minimal model* of Π if $\mathcal{T}, T \models^* \Pi^{\langle \mathcal{T}, T \rangle}$ for every $T \in \mathcal{T}$ [15].

While Def. 4 provides a general definition, its specialisation to $\mathtt{t_f}$ and $\mathtt{t_r}$ is straightforward. When these methods do not result in a distinction, we refer to them by the general name "truth" ($\mathtt{t}$).

Example 5 Consider the EASP-program $\Gamma = \{\mathtt{r}_1, \mathtt{r}_2, \mathtt{r}_3, \mathtt{r}_4\}$ where its rules are explicitly shown below:

$$\mathtt{r}_1 = a \vee b. \qquad \mathtt{r}_2 = c \leftarrow \hat{\mathsf{K}}a \wedge \mathtt{not}\, b. \qquad \mathtt{r}_3 = d \leftarrow \mathtt{not}\,\mathsf{K}a \wedge b. \qquad \mathtt{r}_4 = \bot \leftarrow \mathtt{not}\,\hat{\mathsf{K}}c.$$

We claim that $\{\{a,c\},\{b,d\}\}$ is a $\mathtt{t}$-minimal model of Γ. We first compute the following reducts:

$$\left.\begin{array}{c} a \vee b. \\ c \leftarrow \hat{\mathsf{K}}a \wedge \mathtt{not}\,\bot. \\ d \leftarrow \mathtt{not}\,\bot \wedge b. \\ \bot \leftarrow \mathtt{not}\,\top. \end{array}\right\} \Gamma^{\{\underline{\{a,c\}},\{b,d\}\}} \qquad \text{and} \qquad \left.\begin{array}{c} a \vee b. \\ c \leftarrow \hat{\mathsf{K}}a \wedge \mathtt{not}\,\top. \\ d \leftarrow \mathtt{not}\,\bot \wedge b. \\ \bot \leftarrow \mathtt{not}\,\top. \end{array}\right\} \Gamma^{\{\{a,c\},\underline{\{b,d\}}\}}$$

The above reducts are respectively equivalent to $\{\mathtt{r}_1, c \leftarrow \hat{\mathsf{K}}a, d \leftarrow b\}$ and $\{\mathtt{r}_1, d \leftarrow b\}$: when $\bot$ ($\mathtt{not}\,\top$) appears as a subgoal, the goal fails to hold. This means that the effect of the entire rule $\mathtt{r}$ is negligible, and $\mathtt{r}$ can be safely omitted. When $\top$ ($\mathtt{not}\,\bot$) appears as a subgoal, $\top$ can be dropped from the subgoals of $\mathsf{body}(\mathtt{r})$ as it trivially holds. While $\{\underline{\{a,c\}},\{b,d\}\} \models \Gamma^{\{\underline{\{a,c\}},\{b,d\}\}}$, all its $\mathtt{f}$-weakenings, viz. $\{\underline{\{a\}},\{b,d\}\}$, $\{\underline{\{c\}},\{b,d\}\}$ and $\{\underline{\emptyset},\{b,d\}\}$, refute it. While $\{\{a,c\},\underline{\{b,d\}}\} \models \Gamma^{\{\{a,c\},\underline{\{b,d\}}\}}$, all its $\mathtt{f}$-weakenings, viz. $\{\{a,c\},\underline{\{b\}}\}$, $\{\{a,c\},\underline{\{d\}}\}$ and $\{\{a,c\},\underline{\emptyset}\}$, refute it. Finally, notice that the $\mathsf{S5}$-model $\{\{a,c\}\}$ is the other (unintended) $\mathtt{t_f}$-minimal model of Γ, and both $\mathtt{t}$-minimality tools produce the identical results for Γ.

In a parallel, purely logical context, world-views are alternatively computed as epistemic extensions of equilibrium models. A first step towards epistemic equilibrium logic belongs to Wang&Zhang [19]. As their approach has generalised ES_{94} and also due to page restrictions, we do not include it below.

4 Epistemic Extensions of Equilibrium Logic

Equilibrium logic (EL) is a nonmonotonic formalism, basically proposed by Pearce [11] as a logical and mathematical framework of ASP. EL is based on *here-and-there logic* (HT), a three-valued monotonic logic which is intermediate between classical logic and intuitionistic logic. An HT-model is an ordered pair (H,T) of valuations $H,T \subseteq \mathbb{P}$ satisfying $H \subseteq T$. The semantics of EL, via *equilibrium models*, is obtained through a $\mathtt{t}$-minimality criterion over HT-models: T is an equilibrium model of φ iff $T,T \models_{\mathsf{HT}} \varphi$ (i.e., $T \models \varphi$) and ($\mathtt{t}$-*minimality condition*) $H,T \not\models_{\mathsf{HT}} \varphi$ for any H strictly included in T ($H \subset T$). In summary, Pearce has generalised ASP by characterising its stable-models as equilibrium models in EL.

4.1 Su et al.'s approach (ES_{20a}): autoepistemic equilibrium logic (AEEL)

Inspired by EL's success as a foundational framework for ASP, Su et al. introduced [13, 2, 18] an epistemic extension of EL as an alternative semantics for ES. We here name their approach ES_{20a} and recall how ES_{20a} produces its t-minimal models, namely epistemic equilibrium models (EEMs). For our purposes, we do not include their k-minimality method, selecting ES_{20a}-world-views among its EEMs.

4.1.1 Epistemic here-and-there logic (EHT) and its equilibrium S5-models w.r.t. truth

EHT extends HT with nondual epistemic modalities K and $\hat{\mathsf{K}}$, both of which are primitive and structurally identical to the modalities in EASP. Depending on knowledge-minimality conditions, these modalities may characterise different epistemic concepts, so we do not assign them a literal reading for generality. The language of EHT ($\mathcal{L}_{\mathsf{EHT}}$) is given by the grammar below, where the formulas outside HT are in bold.

$$\varphi \coloneqq p \mid \bot \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \to \varphi \mid \mathbf{K}\varphi \mid \hat{\mathbf{K}}\varphi. \qquad (\text{for } p \in \mathbb{P})$$

As usual, the derived formulas $\neg\varphi$, $\top$, and $\varphi \leftrightarrow \psi$ respectively abbreviate $\varphi \to \bot$, $\bot \to \bot$, and $(\varphi \to \psi) \wedge (\psi \to \varphi)$. A theory is a finite set of formulas. An EASP-program Π is translated to the corresponding EHT-theory Π^* via a map $(.)^*$: given $\Sigma = \{\mathtt{r}_1, \mathtt{r}_2\}$ s.t. $\mathtt{r}_1 = p \vee {\sim} q \leftarrow \hat{\mathsf{K}} r \wedge \mathtt{not}\, s$ and $\mathtt{r}_2 = q \leftarrow \mathtt{not}\,\mathsf{K} p$,

$$\Sigma^* = (\,(\,\hat{\mathsf{K}} r \wedge \neg s\,) \to (\,p \vee \widetilde{q}\,)\,) \;\wedge\; (\,\neg\mathsf{K} p \to q\,) \;\wedge\; \neg(\,q \wedge \widetilde{q}\,).$$

The literal ${\sim}q$ is treated as a new atom $\widetilde{q} \in \mathbb{P}$, and this entails the formula $\neg(q \wedge \widetilde{q})$ to be inserted into Σ^* for consistency purposes. Since it can be easily removed from a logic program with the addition of a constraint $\bot \leftarrow q \wedge \widetilde{q}$ as above, the rest of the paper disregards strong negation $\sim$ for simplicity.

As already mentioned in Ex. 2, the $\hat{\mathsf{K}}$ operator is syntactically different from $\mathsf{M} \in \mathcal{L}_{\mathsf{ES}}$. This is justified by the fact that M is derived as $\mathtt{not}\,\mathsf{K}\,\mathtt{not}$ in ES and so translated into EHT as $\neg\mathsf{K}\neg$ where $\neg$ refers to EHT-negation. Because $\neg\mathsf{K}\neg\varphi$, $\neg\neg\hat{\mathsf{K}}\varphi$, and $\hat{\mathsf{K}}\neg\neg\varphi$ are all equivalent in EHT, the M operator is expected to coincide with $\mathtt{notnot}\hat{\mathsf{K}}$ and $\hat{\mathsf{K}}\mathtt{notnot}$ in a possible extension of EASP-programs to propositional theories, which will be shortly discussed in the next section. Notice that the difference between $\mathsf{M}p$ and $\hat{\mathsf{K}}p$ in EASP resembles that of $\mathtt{notnot}\,p$ and p in ASP. As a result, in an extended language, we expect $\mathsf{M}p$ not to have a world-view, whereas $\{\emptyset, \{p\}\}$ is one easily-understandable world-view for $\hat{\mathsf{K}}p$.

An EHT-model $\langle \mathcal{T}, \mathsf{s} \rangle$ is a refinement of a classical S5-model $\mathcal{T}$ in which valuations $T \in \mathcal{T}$ are replaced by HT-models $(\mathsf{s}(T), T)$ w.r.t. a subset function $\mathsf{s} : \mathcal{T} \to 2^{\mathbb{P}}$, assigning to each $T \in \mathcal{T}$ one of its subsets, i.e., $\mathsf{s}(T) \subseteq T$. Thus, the explicit representation of $\langle \mathcal{T}, \mathsf{s} \rangle$ is given by $\{(\mathsf{s}(T), T)\}_{T \in \mathcal{T}}$. Satisfaction of a formula $\varphi \in \mathcal{L}_{\mathsf{EHT}}$ is defined recursively w.r.t. to the following truth conditions:

$$\begin{array}{lll}
\langle \mathcal{T}, \mathsf{s} \rangle, T \models_{\mathsf{EHT}} p & \text{if} & p \in \mathsf{s}(T); \\
\langle \mathcal{T}, \mathsf{s} \rangle, T \models_{\mathsf{EHT}} \varphi \to \psi & \text{if} & (\langle \mathcal{T}, \mathsf{s} \rangle, T \not\models_{\mathsf{EHT}} \varphi \text{ or } \langle \mathcal{T}, \mathsf{s} \rangle, T \models_{\mathsf{EHT}} \psi) \text{ and} \\
 & & (\langle \mathcal{T}, id \rangle, T \not\models_{\mathsf{EHT}} \varphi \text{ or } \langle \mathcal{T}, id \rangle, T \models_{\mathsf{EHT}} \psi); \\
\langle \mathcal{T}, \mathsf{s} \rangle, T \models_{\mathsf{EHT}} \mathsf{K}\varphi & \text{if} & \langle \mathcal{T}, \mathsf{s} \rangle, T' \models_{\mathsf{EHT}} \varphi \text{ for every } T' \in \mathcal{T}; \\
\langle \mathcal{T}, \mathsf{s} \rangle, T \models_{\mathsf{EHT}} \hat{\mathsf{K}}\varphi & \text{if} & \langle \mathcal{T}, \mathsf{s} \rangle, T' \models_{\mathsf{EHT}} \varphi \text{ for some } T' \in \mathcal{T};
\end{array}$$

where id denotes the identity function. The truth conditions of $\bot$, $\wedge$ and $\vee$ are standard. The EHT-model $\langle \mathcal{T}, id \rangle = \{(T, T)\}_{T \in \mathcal{T}}$ is called *total* and identical to the classical S5-model $\mathcal{T}$. Then, $\mathcal{T}$ is an *equilibrium* S5*-model w.r.t. truth*, or originally an *epistemic equilibrium model* (EEM) of $\varphi \in \mathcal{L}_{\mathsf{EHT}}$ if $\mathcal{T}$ is a classical S5-model of φ, and the following t-minimality condition (referred to as $\mathtt{t}_{\mathcal{F}}$*-minimality*), viz.

$$\text{for every possible subset function } \mathsf{s} \text{ on } \mathcal{T} \text{ with } \mathsf{s} \neq id, \text{ there is } T \in \mathcal{T} \text{ s.t. } \langle \mathcal{T}, \mathsf{s} \rangle, T \not\models_{\mathsf{EHT}} \varphi \qquad (5)$$

holds. $\mathsf{ES}_{\mathsf{20a}}$ further applies a knowledge-minimality ($\mathtt{k}$-minimality) criterion ([18], p. 12), simultaneously functioning two different conditions, upon `EEMs` to determine its world-views, originally referred to as autoepistemic equilibrium models (`AEEMs`). The inspiration comes from autoepistemic logic and the logic of all-that-I-know, and the selection process is carried out by mutual comparison of `EEMs` according to set inclusion and a formula-indexed preorder. Note that applying the same criterion upon EASP's $\mathtt{t_f}$-minimal models to select world-views, we can search for a relationship between two formalisms.

4.2 Cabalar et al.'s approach ($\mathsf{ES}_{\mathsf{20b}}$): founded autoepistemic equilibrium logic (FAEEL)

Cabalar et al. [1] define EHT on a $\hat{\mathsf{K}}$-free fragment of $\mathcal{L}_{\mathsf{EHT}}$. The authors acknowledge that the relation of a second operator (M vs. $\hat{\mathsf{K}}$) to K is under debate, and so they leave its study for future work. Even though not in terms of meaning, the inclusion of $\hat{\mathsf{K}}$ into $\mathsf{ES}_{\mathsf{20b}}$ is methodologically straightforward. Therefore, we here follow the same language $\mathcal{L}_{\mathsf{EHT}}$ for $\mathsf{ES}_{\mathsf{20b}}$ as well in terms of harmony. Moreover, $\mathsf{ES}_{\mathsf{20b}}$ partially contains $\hat{\mathsf{K}}$ when considered in its original language since $\neg\hat{\mathsf{K}}$ and $\hat{\mathsf{K}}\neg$ are EHT-equivalent respectively to $\mathsf{K}\neg$ and $\neg\mathsf{K}$. As a derived formula, M is also included by default in all existing EELs in the form of $\neg\mathsf{K}\neg$. Unlike in $\mathsf{ES}_{\mathsf{20a}}$ where K represents *knowledge*, in this context, $\mathsf{K}\varphi$ reads φ *is one of the agent's beliefs*.

In $\mathsf{ES}_{\mathsf{20b}}$, an EHT-model $\langle\mathcal{T},\mathsf{s_r}\rangle$ is defined w.r.t. a serial subset relation (multi-valued subset function) $\mathsf{s_r}$, relating each $T\in\mathcal{T}$ to at least one element from 2^T, i.e., to some subsets of T. Thus, a serial subset relation $\mathsf{s_r}$ and an S5-model $\mathcal{T}$ give rise to the EHT-model $\mathsf{s_r}=\{(H,T) : T\,\mathsf{s_r}H\}_{T\in\mathcal{T}}$. To illustrate the functional vs. relational nature of the formalisms $\mathsf{ES}_{\mathsf{20a}}$ and $\mathsf{ES}_{\mathsf{20b}}$, take the S5-model $\mathcal{T}=\{T\}$ where $T=\{p,q\}$. Depending on the subset function $\mathtt{s}$ on $\mathcal{T}$, we can only form the EHT-models $\{(\emptyset,T)\}$, $\{(\{p\},T)\}$, $\{(\{q\},T)\}$, and $\{(T,T)\}$ in $\mathsf{ES}_{\mathsf{20a}}$ as we are restricted to choose a unique subset $H=\mathsf{s}(T)$ and so build a unique HT-model (H,T) for each $T\in\mathcal{T}$. However, in EHT_{20}, we can obtain the additional EHT-models

$$\{(\{p\},T),(\{q\},T)\},\quad \{(\emptyset,T),(\{p\},T),(\{q\},T)\},\quad \{(\emptyset,T),(T,T)\},\quad \{(\emptyset,T),(\{p\},T),(T,T)\},\text{ etc.}$$

since as many subset as desired can be chosen for each $T\in\mathcal{T}$, keeping in mind that $\mathsf{s_r}$ is serial.

While the truth conditions are the same, to avoid possible confusion, we recall that $\langle\mathcal{T},\mathsf{s}\rangle,T\models_{\mathsf{EHT}}\varphi$ means $\{(H,T) : H=\mathsf{s}(T)\}_{T\in\mathcal{T}},(H,T)\models_{\mathsf{EHT}}\varphi$ in $\mathsf{ES}_{\mathsf{20a}}$, but here $\mathsf{s_r}(T)$ may refer to more than one subset as $\mathsf{s_r}$ is multi-valued. Thus, we prefer an explicit notation $\{(H,T) : T\,\mathsf{s_r}H\}_{T\in\mathcal{T}},(H,T)\models_{\mathsf{EHT}}\varphi$ to be precise.

An *epistemic equilibrium model* (`EEM`) of $\varphi\in\mathcal{L}_{\mathsf{EHT}}$ is then defined as its classical S5-model $\mathcal{T}$ satisfying a $\mathtt{t_r}$*-minimality* condition: for every multi-valued subset function $\mathsf{s_r}$ on $\mathcal{T}$ s.t. $\mathsf{s_r}\neq id$,

$$\text{there is an HT-model } (H,T) \text{ s.t. } T\,\mathsf{s_r}H \text{ and } \{(H,T) : T\,\mathsf{s_r}H\}_{T\in\mathcal{T}},(H,T)\not\models_{\mathsf{EHT}}\varphi. \tag{6}$$

Once `EEMs` are produced, the next step is to apply Schwarz's [12] minimal model reasoning[1] for nonmonotonic KD45 to select world-views of $\mathsf{ES}_{\mathsf{20b}}$ from among `EEMs`. The operator K obtains its meaning from this approach because in autoepistemic logic, the epistemic operator K characterises the *self-belief* of a rational agent. To weaken a $\mathtt{t_r}$-minimal S5-model (`EEM`) w.r.t belief, $\mathsf{ES}_{\mathsf{20b}}$ needs to generalise `EEMs` to KD45-model structures because minimality w.r.t. belief (b) is tested in nonmonotonic KD45 by examining whether an S5-model has a *preferred* model extension in KD45. To check stability w.r.t. belief in $\mathsf{ES}_{\mathsf{20b}}$, we add a new valuation I into a (candidate) `EEM` $\mathcal{T}$ s.t. $I\notin\mathcal{T}$ and design the resulting KD45 model $\mathcal{T}'=\mathcal{T}\uplus\{I\}$ in a way that I is not accessible by any point in $\mathcal{T}'$ while any point in $\mathcal{T}$ can be accessed by every point in $\mathcal{T}'$. Thus every point in $\mathcal{T}'$, including I, uses the same belief that is determined by $\mathcal{T}$. Formally, $\mathcal{T}'$ is preferred over $\mathcal{T}$, and $\mathcal{T}'$ is a $\mathtt{t_r}$-minimal KD45-model of φ if the following conditions

$$\text{(i)}\ \ \mathcal{T}\uplus\{I\}\,,\,I\models_{\mathsf{KD45}}\varphi \qquad\text{and}\qquad \text{(ii)}\ \ \mathcal{T}\uplus\{(\mathsf{s}(I),I)\}\,,\,(\mathsf{s}(I),I)\not\models_{\mathsf{EHT}}\varphi \tag{7}$$

[1] Schwarz has proved that autoepistemic logic under stable expansions and KD45 under minimal models coincide.

respectively hold. When (7) holds for a candidate $\mathtt{EEM}$ $\mathcal{T}$, this means that $\mathcal{T}$ is not stable (or at equilibrium) w.r.t. belief and fails to be an $\mathtt{AEEM}$ of φ in $\mathsf{ES}_{\mathsf{20b}}$. Notice that the condition (7).(ii) does not require that the points of $\mathcal{T}$ be weakened w.r.t. truth: as $\mathcal{T}$ is an EEM of φ, by definition, any weakeaning of $\mathcal{T}$ results in the formula φ being refuted at some point of $\mathcal{T}$. Also note that due to the KD45-model structure, we weaken I w.r.t. truth in $\mathsf{ES}_{\mathsf{20b}}$ simply by using the subset function s in (7) as $\mathtt{s_r}$ and s provide identical models. We do not reformulate above the details of the method in its original notation as our aim here is to give a brief overview to the reader. However, for our purposes, it is worth mentioning that this b-minimality approach can be easily adapted to $\mathtt{t_r}$-minimal models of EASP as formalised below.

Definition 5 (stable S5-models of EASP w.r.t. truth and belief) Let $\mathcal{T}$ be a nonempty collection of valuations, and let Π be an EASP-program. Then, $\mathcal{T}$ is a *stable* S5-*model* of Π w.r.t. truth and belief if for every $T \in \mathcal{T}$, we have $\mathcal{T}, T \models^*_{\mathtt{r}} \Pi^{\langle \mathcal{T}, T\rangle}$ and for every $I \in 2^{\mathbb{P}} \setminus \mathcal{T}$,

$$\mathcal{T} \uplus \{I\}, I \not\models_{\mathsf{KD45}} \Pi^{\langle \mathcal{T}, I\rangle} \text{ or } \mathcal{T} \uplus \{(\mathsf{s}(I), I)\}, (\mathsf{s}(I), I) \models_{\mathsf{KD45}} \Pi^{\langle \mathcal{T}, I\rangle} \text{ for some subset map } \mathsf{s} \text{ s.t. } \mathsf{s}(I) \subset I. \quad (8)$$

The condition (8) states that $\mathcal{T}$ has no $\mathtt{t_r}$-minimal preferred model in KD45. This definition will then allow us to search for a correspondence between the resulting formalism and $\mathsf{ES}_{\mathsf{20b}}$.

4.3 Su's approach ($\mathsf{ES}_{\mathsf{21}}$): reflexive autoepistemic equilibrium logic (RAEEL)

Su [17] then suggests applying the $\mathtt{k}$-minimality criterion of nonmonotonic SW5 [14] over $\mathtt{EEMs}$ of $\mathsf{ES}_{\mathsf{20a}}$ or $\mathsf{ES}_{\mathsf{20b}}$ to select $\mathtt{AEEMs}$ and proposes $\mathsf{ES}_{\mathsf{21}}$. Remember that the modal logic SW5 is just a reflexive[2] closure of KD45 where K represents *knowledge*. Our underlying intuition is simply because the formulas $\mathsf{K}p$ and $\hat{\mathsf{K}}p$ have respectively the unique AEEMs $\{\{p\}\}$ and $\{\emptyset, \{p\}\}$ in $\mathsf{ES}_{\mathsf{21}}$, regardless of the $\mathtt{t}$-minimality technique chosen, $\mathtt{t_r}$ vs. $\mathtt{t_f}$. While $\hat{\mathsf{K}}p$ has the same AEEM, $\mathsf{K}p$ has no AEEM in $\mathsf{ES}_{\mathsf{20b}}$. In an extended language, $\mathsf{ES}_{\mathsf{94}}$ cannot provide any world views for these formulas, and a slightly modified version $\mathsf{ES}_{\mathsf{11}}$ [6] cannot produce a reasonable model $\{\{p\}\}$ for $\mathsf{K}p \vee q$. These results reinforce the counter-arguments provided in Ex. 1-2 towards their reduct definitions, attacking positive subjective literals. If an atom p can be derived in all stable models of an ASP-program, then the query p? is answered as *true*. Does it provide an enough justification for the derivation of $\mathsf{K}p$? While p has a unique world view $\{\{p\}\}$, why does a stronger expression $\mathsf{K}p$ lack a world-view? Such questions go on... Although it is unclear what researchers intend to capture with K, the above-mentioned EELs, especially $\mathsf{ES}_{\mathsf{20a}}$ and $\mathsf{ES}_{\mathsf{21}}$ with their well-studied minimality tools, are strong formalisms, and in our opinion, they both can serve with their different functionalities (especially towards constraints) for the encoding of different problems.

All existing EELs in the literature employ a twofold world-view computation process. The method is first to compute $\mathtt{t}$-minimal models of a program, upon which a $\mathtt{k}$-minimality criterion is applied. In $\mathsf{ES}_{\mathsf{94}}$, there is no such clear-cut distinction between truth and knowledge minimality conditions; instead, they are given intertwined with each other. The follow-up ES-formalisms are mostly focused on reduct without modifications in the minimality. This makes it difficult to understand the relationships between ES-formalisms proposed in the logic programming domain and the purely logical domain of EL. However, there are some work in the literature, revealing similarities between existing ES-formalisms. For instance, Wang&Zhang [19]) have embedded $\mathsf{ES}_{\mathsf{94}}$ into an EEL they designed; Cabalar et al. [1] have proved that AEEMs of $\mathsf{ES}_{\mathsf{20b}}$ and founded world-views of $\mathsf{ES}_{\mathsf{94}}$ coincide under a foundedness property they proposed. We tackle this research topic in reverse direction by following Ferraris' work, which captures equilibrium models as stable models. To achieve this, we propose a versatile and comprehensive framework called EASP that can evolve into various EELs, incorporating their $\mathtt{k}$-minimality conditions.

[2] Schwarz [12] has proved that reflexive autoepistemic logic and nonmonotonic SW5 coincide under their specific semantics.

Moreover, compared to related work, it is evident how EASP accommodates existing EELs through the traditional nature of EASP. The next section clarifies how we accomplish this in a unifying framework.

5 Correspondence between EASP and EEL

This section first generalises Ferraris' lemma, presented in ([3], p. 3), to S5, KD45 and SW5-models.

Lemma 1 *Given $I = \{1,\dots,n\}$, let $\mathcal{T} = \{T_i\}_{i\in I} = \{T_i \ : \ i \in I\}$ be an* S5*-model, and let $\mathsf{s} : \mathcal{T} \to 2^{\mathbb{P}}$ be a subset function s.t. $\mathsf{s}(T_i) = H_i \subseteq T_i$ for every $i \in I$. For an* EASP*-program Π,*

$$\{H_1,\dots,H_n\}\ ,\ H_j\ \models_{\mathsf{S5}} \Pi^{\langle\mathcal{T},T_j\rangle} \quad \textit{iff} \quad \{(H_i,T_i) : i \in I\}\ ,\ (H_j,T_j)\ \models_{\mathsf{EHT}} \Pi^*, \quad \textit{for every } j \in I.$$

The lemma is proven by structural induction. As $H_i = H_j$ is possible for some $i, j \in I$, we consider $\{H_i\}_{i\in I}$ as a multiset and employ the traditional reduct introduced in Def. 4. Under this general result, we can clearly see how EELs appear in the logic programming domain and vice versa.

We begin with EEMs of ES_{20a}: for an EASP-program Π, let $\mathcal{T} = \{T_i\}_{i\in I} = \{T_1,\dots,T_n\}$ be an EEM of Π^*. By definition of EEM in ES_{20a} (5), we have (1) $\mathcal{T}, T_i \models_{\mathsf{S5}} \Pi^*$ for every $i \in I$ and (2) for every non-identity subset function s on $\mathcal{T}$ s.t. $\mathsf{s}(T_i) = H_i$ for each i, there is $k \in I$ s.t. $\langle\mathcal{T},\mathsf{s}\rangle, T_k \not\models_{\mathsf{EHT}} \Pi^*$. The model $\langle\mathcal{T},\mathsf{s}\rangle$ gives rise to the EHT-model $\{(H_i,T_i)\}_{i=1}^n = \{(H_1,T_1),\dots,(H_n,T_n)\}$, and so, $\{(H_i,T_i)\}_{i=1}^n,(H_k,T_k) \not\models_{\mathsf{EHT}} \Pi^*$. First let $\mathsf{s} = id$ in Lemma 1, then $H_i = T_i$ for each i. Recall that $\langle\mathcal{T}, id\rangle$ refers to the classical S5-model $\mathcal{T}$. The condition (1) so implies $\{T_1 \dots T_n\}, T_j \models_{\mathsf{S5}} \Pi^{\langle\mathcal{T},T_j\rangle}$, for every $j \in I$. Again by Lemma 1, the condition (2) refers to a more relaxed $\mathtt{t_f}$-minimality criterion not discussed in Sect. 3, saying that "for every subset function s with $\mathsf{s} \neq id$, there is $k \in I$ s.t. $\{\mathsf{s}(T_1),\dots,\mathsf{s}(T_n)\}, \mathsf{s}(T_k) \not\models_{\mathsf{S5}} \Pi^{\langle\mathcal{T},T_k\rangle}$". To sum up, we have:

$$\begin{aligned}&\mathcal{T}, T \models_{\mathsf{S5}} \Pi^{\langle\mathcal{T},T\rangle} \text{ for every } T \in \mathcal{T} \quad \text{and} \\ &\text{for every subset function } \mathsf{s} \neq id, \text{ there is } T' \in \mathcal{T} \text{ s.t. } \{\mathsf{s}(T)\}_{T\in\mathcal{T}}, \mathsf{s}(T') \not\models_{\mathsf{S5}} \Pi^{\langle\mathcal{T},T'\rangle}.\end{aligned} \tag{9}$$

Since EHT-models of ES_{20b} are formed in a relational structure, first we should refine Lemma 1.

Lemma 2 *Let Π be an* EASP*-program. Let $\mathcal{T}$ be an* S5*-model, and let $\mathsf{s_r}$ be a multi-valued subset function on $\mathcal{T}$ s.t. $\mathsf{s_r} = \{(H,T) : T\,\mathsf{s_r}H\}_{T\in\mathcal{T}}$. For every $T \in \mathcal{T}$, let H be s.t. $T\,\mathsf{s_r}H$. Then, we have:*

$$\{H : T\,\mathsf{s_r}H\}_{T\in\mathcal{T}}\ ,\ H\ \models_{\mathsf{S5}} \Pi^{\langle\mathcal{T},T\rangle} \quad \textit{iff} \quad \{(H,T) : T\,\mathsf{s_r}H\}_{T\in\mathcal{T}}\ ,\ (H,T)\ \models_{\mathsf{EHT}} \Pi^*.$$

Pursuing a similar proof, we can also capture EEMs of ES_{20b} in EASP. While the line (9) remains the same, we again obtain a more-relaxed $\mathtt{t_r}$-minimality condition compared to one proposed in Sect. 3:

$$\begin{aligned}&\text{for every multi-valued subset function } \mathsf{s_r} \text{ s.t. } \neq id, \\ &\{H : T\,\mathsf{s_r}H\}_{T\in\mathcal{T}}\ ,\ H'\ \not\models_{\mathsf{S5}} \Pi^{\langle\mathcal{T},T'\rangle} \text{ for some } T' \in \mathcal{T} \text{ and for some } H' \text{ s.t. } T'\mathsf{s_r}H'.\end{aligned} \tag{10}$$

The extensions of Lemma 1-2 to KD45 and SW5-model structures and reflecting generalised EEMs in such weaker model structures to EASP are straightforward. We now perform the same task in the opposite direction and embed Def. 4 into EEL domain. Using Lemma 1, $\mathcal{T} = \{T_i\}_{i\in I}$ is a $\mathtt{t_f}$-minimal S5-model of Π iff $\mathcal{T}, T_i \models_{\mathsf{S5}} \Pi^*$ for every $i \in I$ and for every $j \in I$, we have

$$\text{for every subset map } \mathsf{s} \text{ s.t. } \mathsf{s}|_{\mathcal{T}\setminus\{T_j\}} = id \text{ and } \mathsf{s}(T_j) \subset T_j,\ \{(\mathsf{s}(T_i),T_i)\}_i,(\mathsf{s}(T_j),T_j) \not\models_{\mathsf{EHT}} \Pi^*. \tag{11}$$

We leave it to the reader to generalise this result to $\mathtt{t_r}$-minimal S5-models of EASP by Lemma 2.

Through the same approach, we try to analyse ES_{94}-semantics: let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a world-view of an ES-program Π. By definition, $\mathcal{T}$ is the maximal set w.r.t. subset relation $\subseteq$ satisfying (1) $\mathcal{T} \models \Pi^{\mathcal{T}}$ and (2) $(\mathcal{T} \setminus \{T\}) \cup \{H\}, H \not\models \Pi^{\mathcal{T}}$ for every $H \subset T$, for every $T \in \mathcal{T}$. Notice that $\Pi^{\mathcal{T}} = \Pi^{\langle \mathcal{T}, T_i \rangle}$ for every $i \in I$ as ES_{94}-reduct eliminates only extended subjective literals. This definition, except maximality condition, coincides with $\mathtt{t_f}$-minimal $\mathsf{S5}$-model definition of EASP, and so with (11) by Lemma 1. However, $\{\{p\}\}$ is a world-view of $p \leftarrow \mathsf{K}p$, but not a $\mathtt{t_f}$-minimal $\mathsf{S5}$-model of Π. For some reasons, we cannot apply Ferraris' generalised lemma (i.e., Lemma 1) to ES_{94}-semantics.

6 Conclusion

In this paper, we first discuss the problems that arise under Gelfond's original ES_{94}-semantics, aiming to shed light on the underlying reasons for these issues. We also briefly overview the follow-up semantics, that were primarily proposed to address the limitations of ES_{94}. Next, we introduce a flexible and robust framework for epistemic logic programs called EASP, which already accommodates Su's traditional $\mathtt{t_f}$-minimal $\mathsf{S5}$-models, as studied in [15], and their novel variations known as $\mathtt{t_r}$-minimal $\mathsf{S5}$-models.

We recognise that all existing epistemic equilibrium logics (EELs) in the literature share a two-step world-view computation process. This motivates us to explore their similarities and beyond within the EASP context. To this end, we generalise Ferraris' lemma (see [3], p. 3), which establishes a correlation between stable models and equilibrium models, to the epistemic case. We then examine how these EELs are reflected within the EASP framework. This approach also allows us to investigate whether different $\mathtt{t_f}$ ($\mathtt{t_r}$) minimality methods, such as those presented in [15] and [18], produce the same results when considered at least within the current EASP language fragment. It is worth noting that the technique in [15] is slightly easier than that in [18], which raises an immediate research question for future studies. Furthermore, Ferraris' generalised lemmas lead to the strong equivalence characterisations of EASP-programs through the logical equivalences of their translations in EHT, akin to Lifschitz et al.'s finding [10] in regular ASP. Finally, future work will also involve a more detailed investigation of how Gelfond's ES_{94}-semantics can be reflected into the EEL domain, following a similar approach as discussed in this paper. This study will help us better identify the problems of ES_{94}, as well as its possible similarities with other ES-semantic approaches originally proposed in the EEL domain.

References

[1] Pedro Cabalar, Jorge Fandinno & Luis Fariñas del Cero (2020): *Autoepistemic answer set programming*. *Artif. Intell.* 289, p. 103382. Available at `https://doi.org/10.1016/j.artint.2020.103382`.

[2] Luis Fariñas del Cerro, Andreas Herzig & Ezgi Iraz Su (2015): *Epistemic Equilibrium Logic*. In Qiang Yang & Michael J. Wooldridge, editors: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, AAAI Press, pp. 2964–2970. Available at `http://ijcai.org/Abstract/15/419`.

[3] Paolo Ferraris (2005): *Answer Sets for Propositional Theories*. In Chitta Baral, Gianluigi Greco, Nicola Leone & Giorgio Terracina, editors: *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5-8, 2005, Proceedings*, *Lecture Notes in Computer Science* 3662, Springer, pp. 119–131. Available at `https://doi.org/10.1007/11546207_10`.

[4] Michael Gelfond (1991): *Strong Introspection*. In Thomas L. Dean & Kathleen R. McKeown, editors: *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 1*, AAAI Press / The MIT Press, pp. 386–391. Available at `http://www.aaai.org/Library/AAAI/1991/aaai91-060.php`.

[5] Michael Gelfond (1994): *Logic Programming and Reasoning with Incomplete Information*. *Ann. Math. Artif. Intell.* 12(1-2), pp. 89–116. Available at https://doi.org/10.1007/BF01530762.

[6] Michael Gelfond (2011): *New Semantics for Epistemic Specifications*. In James P. Delgrande & Wolfgang Faber, editors: *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, *Lecture Notes in Computer Science* 6645, Springer, pp. 260–265. Available at https://doi.org/10.1007/978-3-642-20895-9.

[7] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In Robert A. Kowalski & Kenneth A. Bowen, editors: *Logic Programming, Proceedings of the 5th International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*, MIT Press, pp. 1070–1080.

[8] Michael Gelfond & Vladimir Lifschitz (1991): *Classical Negation in Logic Programs and Disjunctive Databases*. *New Gener. Comput.* 9(3/4), pp. 365–386. Available at https://doi.org/10.1007/BF03037169.

[9] Patrick Thor Kahl (2014): *Refining the semantics for epistemic logic programs*. Ph.D. thesis, Texas Tech University, Department of Computer Science, Lubblock, TX, USA.

[10] Vladimir Lifschitz, David Pearce & Agustín Valverde (2001): *Strongly equivalent logic programs*. *ACM Trans. Comput. Log.* 2(4), pp. 526–541. Available at https://doi.org/10.1145/383779.383783.

[11] David Pearce (2006): *Equilibrium logic*. *Ann. Math. Artif. Intell.* 47(1-2), pp. 3–41, doi:10.1007/s10472-006-9028-z.

[12] Grigori Schwarz (1992): *Minimal Model Semantics for Nonmonotonic Modal Logics*. In: *Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS) '92, Santa Cruz, California, USA, June 22-25, 1992*, pp. 34–43, doi:10.1109/LICS.1992.185517.

[13] Ezgi Iraz Su (2015): *Extensions of equilibrium logic by modal concepts. (Extensions de la logique d'équilibre par des concepts modaux)*. Ph.D. thesis, Institut de Recherche en Informatique de Toulouse, France. Available at https://tel.archives-ouvertes.fr/tel-01636791.

[14] Ezgi Iraz Su (2017): *A Monotonic View on Reflexive Autoepistemic Reasoning*. In Marcello Balduccini & Tomi Janhunen, editors: *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, *Lecture Notes in Computer Science* 10377, Springer, pp. 85–100. Available at https://doi.org/10.1007/978-3-319-61660-5_10.

[15] Ezgi Iraz Su (2019): *Epistemic Answer Set Programming*. In Francesco Calimeri, Nicola Leone & Marco Manna, editors: *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, *Lecture Notes in Computer Science* 11468, Springer, pp. 608–626. Available at https://doi.org/10.1007/978-3-030-19570-0_40.

[16] Ezgi Iraz Su (2020): *A Unifying Approach for Nonmonotonic S4F, (Reflexive) Autoepistemic Logic, and Answer Set Programming*. *Fundam. Informaticae* 176(3-4), pp. 205–234. Available at https://doi.org/10.3233/FI-2020-1972.

[17] Ezgi Iraz Su (2021): *Refining the Semantics of Epistemic Specifications*. In Andrea Formisano, Yanhong Annie Liu, Bart Bogaerts, Alex Brik, Verónica Dahl, Carmine Dodaro, Paul Fodor, Gian Luca Pozzato, Joost Vennekens & Neng-Fa Zhou, editors: *Proceedings 37th International Conference on Logic Programming, ICLP Technical Communications 2021, Porto (virtual event), 20-27th September 2021*, *EPTCS* 345, pp. 113–126. Available at https://doi.org/10.4204/EPTCS.345.25.

[18] Ezgi Iraz Su, Luis Fariñas del Cerro & Andreas Herzig (2020): *Autoepistemic equilibrium logic and epistemic specifications*. *Artif. Intell.* 282, p. 103249. Available at https://doi.org/10.1016/j.artint.2020.103249.

[19] Kewen Wang & Yan Zhang (2005): *Nested Epistemic Logic Programs*. In Chitta Baral, Gianluigi Greco, Nicola Leone & Giorgio Terracina, editors: *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5-8, 2005, Proceedings*, *Lecture Notes in Computer Science* 3662, Springer, pp. 279–290. Available at https://doi.org/10.1007/11546207_22.

ASP-driven User-interaction with *Clinguin*

Alexander Beiser Susana Hahn Torsten Schaub

University of Potsdam, Germany Potassco Solutions, Germany

alexander.beiser@tuwien.ac.at hahnmartinlu@uni-potsdam.de torsten@uni-potsdam.de

We present *clinguin*, a system for ASP-driven user interface design. *Clinguin* streamlines the development of user interfaces for ASP developers by letting them build interactive prototypes directly in ASP, eliminating the need for separate frontend languages. To this end, *clinguin* uses a few dedicated predicates to define user interfaces and the treatment of user-triggered events. This simple design greatly facilitates the specification of user interactions with an ASP system, in our case *clingo*.

1 Introduction

The growing popularity of Answer Set Programming (ASP; [13]) in both academia and industry necessitates the development of user-friendly graphical interfaces to cater to end users. This is especially critical for interactive applications where users engage in iterative feedback loops with ASP systems. Examples include timetabling or product configuration tools. This leads to challenges in frontend development and requires skills in areas beyond ASP development. In addition, custom solutions have a limited reach, as they cannot be easily adapted.

Clinguin addresses this challenge and streamlines User Interface (UI) development for ASP developers by letting them build interactive prototypes directly in ASP, eliminating the need for separate frontend languages. To this end, *clinguin* uses a few dedicated predicates to define UIs and the treatment of user-triggered events. This simple design greatly facilitates the specification of user interactions with an ASP system, in our case *clingo* [12]. Our approach shares similarities with the ASP-driven visualization system *clingraph* [10]. In fact, *clinguin* can be regarded as the interactive counterpart of *clingraph*, whose single-shot approach lacks any interaction capabilities.

In what follows, we rely on a basic acquaintance with ASP, the ASP system *clingo* [7], and some rudimentary Python knowledge. We explain specialized concepts as they are introduced throughout the text.

2 Architecture and Workflow of *clinguin*

Clinguin uses a client-server architecture, where communication occurs via the HTTP protocol (RESTful[1]). JSON is used for message content between client and server. The server leverages *clingo* (5.7) and calculates the information needed to build the UI; this is used by the client to render the corresponding UI and update it based on the user's interaction. The update is either handled directly at the client-side or sent back to the server for further processing, triggering a new interaction loop.

Figure 1 illustrates the architecture and workflow of *clinguin*. It focuses on the key technical components: server, client, and user interface (UI). Furthermore, it shows how the UI is generated and how user interactions are handled within the system.

[1] `https://en.wikipedia.org/wiki/REST`

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 215–228, doi:10.4204/EPTCS.416.19

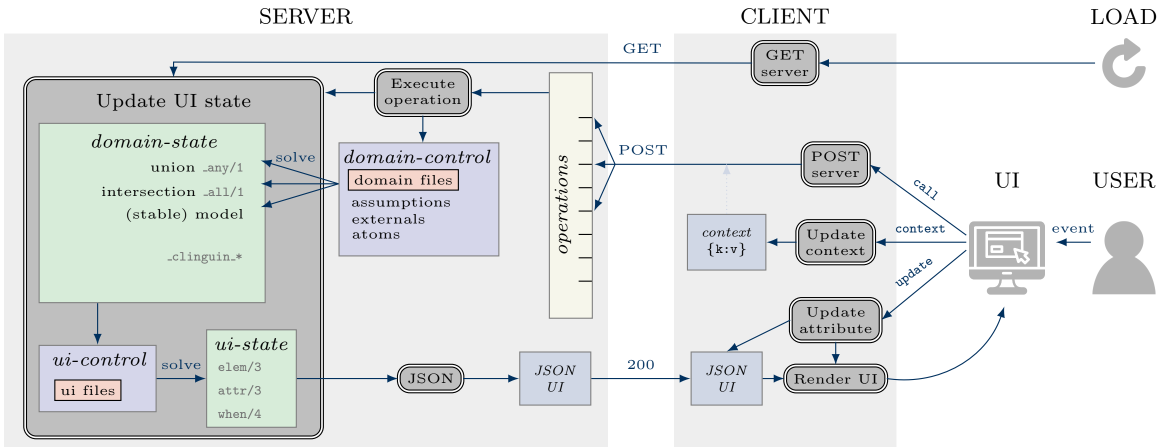


Figure 1: Architecture and Workflow of *clinguin*

In what follows, we illustrate *clinguin*'s approach with a detailed, step-by-step walkthrough using a simple running example about assigning people to seats where a cat person and a dog person cannot share the same table. The corresponding problem instance and encoding are given in Listings 1 and 2, respectively. We refer to them as the *domain files* among the input files; they are highlighted in pink in Figure 1.

```
person("Susana",cat). person("Alexander",cat). person("Torsten",dog).
seat((1,1..2)). seat((2,1..3)).
```

Listing 1: A simple problem instance (`ins.lp`)

```
1  {assign(P,S):seat(S)}:- person(P,_).

3  :- person(P,_), #count{S:assign(P, S)}!=1, cons(exacly_one, _).
4  :- assign(P,S), assign(P',S), P'>P, cons(only_one, _).
5  :- assign(P,(T,_)), assign(P',(T,_)), person(P,cat), person(P',dog),
6     cons(different_type, _).

8  cons(exacly_one, "All people need exactly one seat").
9  cons(only_one, "Two people can not be seated on the same seat").
10 cons(different_type, "All people on a table must prefer the same pet").
```

Listing 2: A simple problem encoding (`enc.lp`)

The choice in Line 1 of Listing 2 generates possible assignments of a person p to a seat s. Each seat s is defined by a pair (t,c) of a numbered table t and chair c. Lines 3 to 6 constrain the assignment, ensuring that a person is assigned to exactly one seat where everyone at the table shares the same pet preference. Each of these integrity constraints is conditioned with an instance of `cons/2` defined as facts in Lines 8 to 10. The first argument of `cons/2` identifies the constraint violation, while the second argument provides the corresponding user-friendly explanation. Although the introduction of these atoms may appear unnecessary at this point, they will be used to identify constraint violations in Section 3.

The server is started with the domain and UI files as command-line arguments:

```
clinguin server --domain-files ins.lp enc.lp --ui-files ui-tables.lp
```

Using the domain files, the server creates the *clingo* object *domain-control* (in purple in Figure 1), which employs multi-shot solving by default. The other input files, called *ui-files* and given in pink in Figure 1, are used to generate a single stable model composed of atoms defining the layout, style, and functionality of the interface, collectively forming the *ui-state*. To this end, another *clingo* object, *ui-control* in purple in Figure 1, is restarted on each update.

Upon launching the client with '`clinguin client`', it requests the UI state (or *ui-state*) from the server (see the *GET* arrow in Figure 1). Once received, the client utilizes a front-end language to render the corresponding user interface. We use *Angular*[2] as front-end language with *Bootstrap*[3] as a toolkit for customization. Subsequent user interactions with the UI (usually) generate new requests to the server, providing details about the selected operations. These operations are predefined by the server and allow users to interact with the *domain-control* in different ways. Examples include adding a user selection as an assumption to the solver, setting the value of an external atom, or obtaining the next solution[4]. Once the server completes the selected operations, it constructs a hierarchical JSON structure of the updated *ui-state* and returns it to the client for rendering.

The *ui-state* is defined by predicates `elem/3`, `attr/3` and `when/4`, for specifying the UI's layout, style and functionality, respectively (see green box with header *ui-state* in Figure 1). The corresponding atoms are mapped into Python classes using *clorm*[5], a Python library that provides an object-relational mapping interface to *clingo*. An atom `elem(X,T,X')` defines an element `X` of type `T` inside element `X'`. Such UI *elements* are the visual representations of objects or features in an interface, such as a button, text field, dropdown menu, and more[4]. The *attributes* of an element, such as position and style, are specified by `attr(X,K,V)`, where `X` is an element, and `K` and `V` denote the attribute's name (key) and value, respectively.

The reactive behavior of the UI is defined by atoms of form `when(X,E,A,P)`, which can be interpreted as expressing: "When event `E` is triggered on element `X`, it is followed by an action `A` with arguments `P`". An *event* refers to an action initiated by the user, such as clicking, double-clicking, or entering text. An *action* is a system response triggered by the UI event. This action must be one of the following:

- a `call` to the server (POST operation), where `P` represents one or multiple server operations,
- a local attribute `update`, where `P` is a triple `(X',K,V)` leading to an update of attribute `K` on element `X'` by `V`,
- a local update to a `context` defined as a dictionary, where `P` is the key-value pair to update and keep as local memory.

The atoms constituting the *ui-state* are generated from the encodings provided as ui-files along with facts describing the *domain-state* (see green box with the identical header in Figure 1). The latter state provides valuable insights into the current state of the *domain-control* object, including the intersection and union of the stable models generated by the *domain-control*, a single stable model for focused exploration, and internal information encapsulated in atoms whose predicates typically start with `_clinguin_`. Collectively, these facts represent relevant information for generating the UI.

Let us illustrate this with our example. Listing 3 shows the ui-file generating the UI snapshots in Figure 2. Line 1 creates a window element labeled `w` and places it inside the overarching root element. Line 2 adds an attribute to window `w` stating that its children elements form a row. Similarly, Line 4 creates a container `tables` and places it inside window `w`. This container groups all elements representing tables. Accordingly, Line 6 defines a container for each table `table(T)` with number `T`. The table numbers are

[2] `https://angular.io`

[3] `https://getbootstrap.com`

[4] For the full list of supported elements and operations, we refer the reader to the system's documentation at `https://clinguin.readthedocs.io/en/latest`.

[5] `https://github.com/potassco/clorm`

```
1  elem(w, window, root).
2  attr(w, flex_direction, row).

4  elem(tables, container, w).

6  elem(table(T), container, tables):- seat((T,_)).
7  attr(table(T), order, T):- seat((T,_)).
8  attr(table(T), width, 200):- seat((T,_)).
9  attr(table(T), class, ("bg-primary";"bg-opacity-25";"rounded";
10                          "d-flex";"flex-column";"align-items-start";
11                          "p-2";"m-2"
12                        )):- seat((T,_)).

14 elem(table_label(T), label, table(T)):- seat((T,_)).
15 attr(table_label(T), order, 1):- seat((T,_)).
16 attr(table_label(T), label, @concat("Table",T)):- seat((T,_)).

18 elem(seat_dd((T,C)), dropdown_menu, table(T)):- seat((T,C)).
19 attr(seat_dd((T,C)), order, C+1):- seat((T,C)).
20 attr(seat_dd(S), class, ("btn-sm";"btn-primary";"m-2")):- seat(S).
21 attr(seat_dd(S), selected, P):- _all(assign(P,S)).

23 elem(seat_ddi(S,P), dropdown_menu_item, seat_dd(S)):-_any(assign(P,S)).
24 attr(seat_ddi(S,P), label, P):- _any(assign(P,S)).
25 when(seat_ddi(S,P), click, call, add_assumption(assign(P,S), true)):-
26                        _any(assign(P,S)).
```

Listing 3: An encoding for table handling (`ui-tables.lp`)

drawn from the instance using atoms with predicate `seat/1` (as seen in Listing 1). Line 7 uses attribute `order` to order tables based on their number. Similarly, Line 8 sets the width of these elements.

In HTML, the *class* attribute specifies one or more class names for an element. These class names correspond to styles defined in a style sheet, which determines the visual properties of the element. *Bootstrap* provides a predefined set of classes that help ensure consistent styling across a user interface. As common in UI design, these classes draw from a custom color palette. For *clinguin*, we crafted this color palette with primary (blue) and secondary (purple) colors as well as special colors representing information, warnings and errors. Lines 9 to 12 set eight of such *Bootstrap* `class` names for each table element. This is done via *clingo*'s pooling operation '`;`' to expand rules. In detail, the `class` names in Line 9 address the background color, opacity and rounded corners. Line 10 deals with the orientation of all elements in the container. And Line 11 addresses padding and margin.

Lines 14 to 16 create a label element with the table number as title in the first position of the table container. To facilitate this, *clinguin* provides several external Python functions, such as `@concat` for label assembly Line 16.

So far, we only used ASP to generate a set of facts capturing static aspects of a user interface. Next, we want to leverage ASP to present users with a well-defined set of choices for selection. To this end, we differentiate between necessary and possible selections in view of what is already chosen by the user. These selections can be captured via the intersection and union of the stable models of the domain files, while also incorporating the user's selections. A necessary selection belongs to all stable models, and a possible one to at least one. In technical terms, this is achieved by manipulating and reasoning with the ASP system *clingo* encapsulated within the *domain-control* object. Recall that the latter is initialized with the domain files. User selections can alternatively be incorporated into the *domain-control* object in terms of assumptions, externals, or regular atoms (cf. [12]). To further include atoms belonging to the intersection and union of the stable models, we reify them via predicates `_all/1` and `_any/1`, respectively, and add the resulting atoms to the current *domain-state*. This is accomplished by two consecutive invocations to the current *domain-control* object, once setting *clingo* option `-enum-mode` to `cautious` and then to `brave`. Although predicates `_all/1` and `_any/1` resemble epistemic operators, their usage is restricted to passing information from the domain to the UI side.

Now, let us explore how *clinguin* manages this in our running example. Lines 18 to 21 define a `dropdown_menu` for each seat and add it to the corresponding `table` container. The heading text of each menu is defined with attribute `selected`. We use this attribute to indicate necessary selections in Line 21. Only if a person `P` is assigned the same seat `S` in all stable models, expressed by `_all(assign(P,S))`, its name is shown as the respective menu text. Similarly, Lines 23 to 25 define all possible seat selections by `dropdown_menu_item`[s] in terms of `_any(assign(P,S))`.

Line 25 dictates the actions occurring when a user `click`[s] on an item within the dropdown menu. If so, a `call` action is initiated and transmitted to the server. In our case, we model user selections as assumptions. Accordingly, server operation `add_assumption` is invoked with arguments `assign(p,s)` and `true`, reflecting the user's selection of person `p` at seat `s`. This operation results in the addition of atom `assign(p,s)` to the *domain-control* object as a true assumption.[6]

Semantically, this amounts to adding the integrity constraint '`:- not assign(p,s).`' and thus forcing the domain encoding to infer `assign(p,s)`. All this is reflected by the upper path from the UI to *domain-control* in Figure 1.

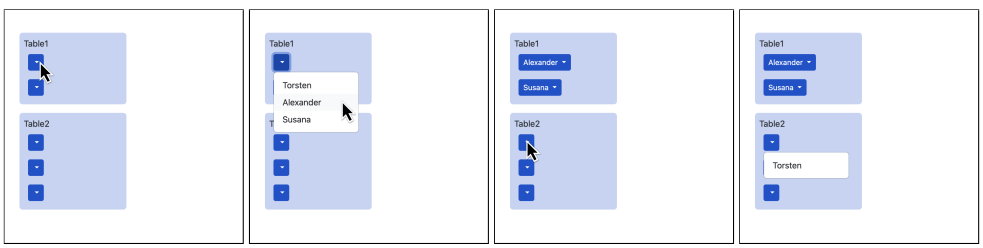


Figure 2: User interaction via mouse actions using ui-file `ui-tables.lp`.

The second screenshot in Figure 2 shows that initially all persons are possible selections for the first seat at Table 1. Once Alexander's seat is chosen, *clinguin* adds the corresponding assumption to the *domain-control* object. This automatically leads to Susana's assignment to the second seat, since all resulting stable models agree on this. Moreover, all three seats at Table 2 have now only the single option Torsten left, since none of the remaining stable models seats Alexander or Susana at this table.

For further illustration, we now extend the functionality of our example by solution browsing. That is, we add Listing 4 to Listing 3 and continue the user interaction from Figure 2 in Figure 3. To begin with, Lines 1 to 3 create a `menu_bar` along with a `title` and an `icon`.[7] Lines 5 to 8 add a `button` to the `menu_bar` to iterate through solutions. When the button is clicked, the server operation `next_solution` is called. This operation computes a new stable model for exploration and adds it to the *domain-state*.

While the previous part of the UI encoding fixes static aspects of the user interface, we now turn to the dynamic part. Our design of solution browsing revolves around displaying the choices from the

[6]As discussed below, assumptions are recorded via predicate `_clinguin_assume/2` in the *domain-state*.

[7]For handling icons, we use the icon library *Font Awesome* `https://fontawesome.com`.

```
1  elem(menu_bar, menu_bar, w).
2  attr(menu_bar, title, "Table placement").
3  attr(menu_bar, icon, "fa-utensils").

5  elem(menu_bar_next, button, menu_bar).
6  attr(menu_bar_next, label, "Next").
7  attr(menu_bar_next, icon, "fa-forward-step").
8  when(menu_bar_next, click, call, next_solution).

10 attr(seat_dd(S), selected, P):- assign(P,S), _clinguin_browsing.
11 attr(seat_dd(S), class, "text-success"):- _clinguin_browsing,
12                              assign(P,S), not _all(assign(P,S)).
13 attr(seat_dd(S), class, "opacity-75"):- _all(assign(P,S)),
14                              not _clinguin_assume(assign(P,S), true).
```

Listing 4: An encoding for menu handling (`ui-menu.lp`)

current stable model stored in the *domain-state*. This approach allows us to differentiate between three types of selections: user selections, derived necessary selections, on which all stable models agree, and choices from the stable model under exploration. To control this behavior, we rely upon the atom `_clinguin_browsing`, whose presence in the *domain-state* indicates whether *clinguin* is in browsing mode. Recall that we show necessary selections in the UI by setting the `selected` attribute of the dropdown menu (in Line 21 of Listing 3). The same attribute is used in Line 10 to visualize choices of the stable model at hand, when in browsing mode. To visually distinguish the different choices, we reduce the opacity of derived necessary choices and display non-necessary choices from the stable model at hand in green text; user selections remain unchanged. This is done in Lines 11 to 14. The first rule displays a seat assignment in green text, viz. `"text-success"`, if it belongs to the explored but not all stable models. The second rule reduces the opacity of seat selections, which belong to all stable models without being enforced by a user selection. Since we represent user choices as assumptions, we can easily determine if a seat selection originated from the user by verifying whether it was explicitly assumed or not. The fact that *clinguin* records all current assumptions in the *domain-state* via predicate `_clinguin_assume/2` in their reified form explains the condition in Line 14.

Figure 3 shows the effects of user interactions after performing the actions from Figure 2. Unlike

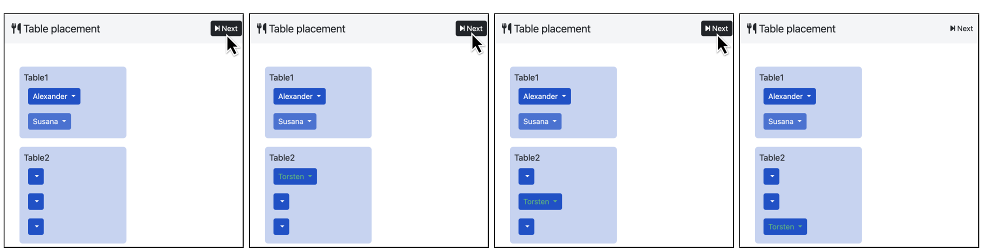


Figure 3: User interaction via mouse actions using UI files `ui-tables.lp` and `ui-menu.lp`.

above, the opacity of the dropdown for the second seat of Table 1 is now reduced, indicating that this information is derived and not selected. The user then clicks the Next button three times to iterate through all possible solutions. These solutions show the three alternative seat assignments for Torsten in green.

Let us take our example a step further and explore how *clinguin* incorporates user input using *modals*, that is, UI elements that appear on top of the UI and temporarily deactivate the underlying content.

```
1  elem(people, container, w).

3  elem(person(P), button, people):- person(P,_).
4  attr(person(P), label, P):- person(P,_).
5  attr(person(P), class, ("disabled";"m-2";"btn-sm")):- person(P,_).
6  attr(person(P), class, ("btn-outline-secondary")):- person(P,cat).
7  attr(person(P), class, ("btn-outline-warning")):- person(P,dog).
8  attr(person(P), icon, @concat("fa-",O)):- person(P,O).

10 elem(add_person, button, people).
11 attr(add_person, label, "Add person").
12 attr(add_person, icon, "fa-user-plus").
13 attr(add_person, class, ("btn-info";"m-2")).
14 when(add_person, click, update, (add_modal, visibility, shown)).

16 elem(add_modal, modal, w).
17 attr(add_modal, title, "Add person").
18 elem(modal_content, container, add_modal).
19 attr(modal_content, class, ("d-flex";"flex-column")).

21 elem(name_tf, textfield, modal_content).
22 attr(name_tf, placeholder, "Enter the name").
23 attr(name_tf, order, 1).
24 attr(name_tf, width, 250).
25 when(name_tf, input, context, (name, _value)).

27 elem(btns_container, container, modal_content).
28 attr(btns_container, class, ("d-flex";"flex-row";"justify-content-end")).
29 attr(btns_container, order, 2).

31 pet(cat;dog).
32 elem(add_btn(O), button, btns_container):- pet(O).
33 attr(add_btn(O), label, "Add"):- pet(O).
34 attr(add_btn(cat), class, ("m-1";"btn-secondary";"ml-auto")):- pet(O).
35 attr(add_btn(dog), class, ("m-1";"btn-warning";"ml-auto")):- pet(O).
36 attr(add_btn(O), icon, @concat("fa-",O)):- pet(O).
37 when(add_btn(O), click, context, (pet, O)):- pet(O).
38 when(add_btn(O), click, call,
39      add_atom(person(_context_value(name,str), _context_value(pet)))):- pet(O).
```

Listing 5: An encoding for instance generation (`ui-people.lp`)

More precisely, we extend our UI encoding with Listing 5 to enable users to interactively add people to the problem instance, as showcased in Figure 4.

To begin with, we create in Line 1 a container to group all people in the instance. In Lines 3 to 8 each person is represented as a disabled button, whose color depends on the preferred pet. We use buttons rather than labels to leverage on the icon functionality for representing pets.

Lines 10 to 13 define the button for adding new persons to the instance. Once this button is clicked, Line 14 prescribes an `update` operation that pops up the modal element by setting its `visibility` attribute to `shown` (see the second snapshot in Figure 5). As shown in Figure 1, this action is performed locally in the client without calling the server.

The actual modal element and its main container are defined in Lines 16 to 19. The first element in the modal is a textfield for user input; it is defined in Lines 21 to 25. Whenever a user types an `input` in the `textfield`, Line 25 triggers the action `context`, which saves it in the dictionary of the same name in the client (see Figure 1). The two parameters indicate that the key `name` is assigned the value of the input held by placeholder `_value`. Essentially, the user's input is assigned to a specific key within the dictionary. The actual *context* dictionary is stored locally on the client. Any further user input in the textfield updates the value associated with the same key in the *context* dictionary, effectively replacing the previous input. The entire *context* dictionary is sent to the server along with every `call` action triggered by the user. Once the server sends a response, the *context* dictionary is cleared on the client-side, potentially to prepare for new interactions.

Lines 27 to 29 create a container for two additional buttons. They are defined in Lines 31 to 38, one for adding cat persons and another for adding dog persons. Most interesting is the reactivity of the buttons specified in Lines 37 and 38. First of all, we note that a single event can trigger several actions. Among them, local actions are executed before `call` actions.[8] The action in Line 37 is a local one and adds the selected `pet` to the *context* dictionary, as described above for the `name` entry. Unlike this, Lines 38 and 39 initiate a server `call` adding atoms representing new persons, using predicate `person/2`. The arguments for the atom are drawn from the *context* dictionary. This is accomplished via the lookup function `_context_value(K)` which yields the dictionary value for the key `K`. This replacement is done in the client and thus makes sure that the values are present before calling the server. To provide further validation tools, this lookup function allows for two optional arguments indicating the expected type `T` and a default value `D`, `_context_value(K,T,D)`. Possible values for types are `str`, `int` and `const` for strings, integers and terms, respectively. By including a default value the presence of a value becomes optional.

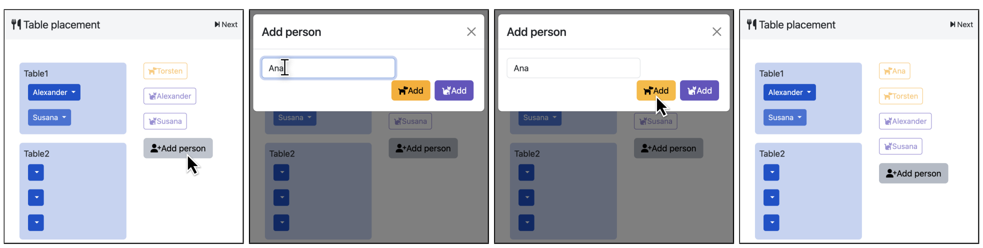


Figure 4: User interaction via mouse actions using `ui-tables.lp`, `ui-menu.lp`, `ui-people.lp`.

Figure 5 shows how a new person can be added by a few clicks. A click on the button 'Add person' changes the visibility of the modal so that it appears in the window. The user then types the person's name, viz. Ana, and clicks on the left button to add Ana as a dog person. The last screenshot shows Ana as part of the people. This is internally reflected by the addition of atom `person("Ana",dog)` to the *domain-control* object.

While the above focuses on instance generation, it's worth mentioning that *clinguin* also allows users to download interactively entered problem instances.

3 Extensibility

Clinguin offers an open modular design that is easily extensible. This allows us to implement extensions without changing the overall workflow. On the server side, this is done by modifying the *backend* component, which provides a layer between the bare ASP solver and the user. To achieve this extensibility,

[8]When multiple `call` actions are triggered by the same event, an order on the corresponding operations can be imposed by including them in a tuple as the last argument of a single `when` atom, reflecting the desired order of execution.

we abstract the responsibilities of backends into different key sections of *clinguin*'s workflow that can be customized: the set of callable operations, the way the control object is handled (solving, grounding, model handling, etc.), the atoms included in the *domain-state*, the way the UI is updated, and the options passed when starting the server. Currently, *clinguin* offers a default backend using *clingo*, which implements all main functionalities for single- and multi-shot solving [8] via *clingo*'s API. Additionally, it includes specialized backends for *clingo*[DL], a *clingo* extension with difference constraints [11], *clingraph*, a *clingo* extension with visualization capabilities [10], and last but not least an explanation backend, which we detail below. On the client side, the *frontend* component can be exchanged to accommodate different GUIs. Currently, *clinguin* offers the web-based frontend *Angular*, used in the paper at hand, and *tkinter*, the standard Python interface to the *Tcl/Tk GUI* toolkit. Alternative GUI frameworks are easily incorporated, mainly because the client-server communication of *clinguin* is standardized by a JSON representation.

3.1 Case study: A backend adding explanations

This section focuses on extending *clinguin*'s backend to provide users with improved feedback when their choices lead to an unsatisfiable scenario. The first part involves pinpointing specific conflicts within the user's selections that impede finding a solution. The second one builds upon this by focusing on presenting error messages tailored to the specific reasons for unsatisfiability.

```
1 elem(seat_ddi(S,P), dropdown_menu_item, seat_dd(S)):-
2                          not _any(assign(P,S)), person(P,_), seat(S).
3 attr(seat_ddi(S,P), label, P):-
4                          not _any(assign(P,S)), person(P,_), seat(S).
5 when(seat_ddi(S,P), click, call, add_assumption(assign(P,S), true)):-
6                          not _any(assign(P,S)), person(P,_), seat(S).
7 attr(seat_ddi(S,P), class, "text-danger"):-
8                          not _any(assign(P,S)), person(P,_), seat(S).
9 attr(seat_dd(S), class, ("text-danger")):- _clinguin_mus(assign(P,S)).
```

Listing 6: UI to handle basic explanations (`ui-explain.lp`)

We start by revealing all options, including those not leading to any solution. This is done in Listing 6 by exclusively using features explained above. The encoding adds all infeasible options in Lines 1 to 6 and informs the user that they belong to no solution in Line 7 (ignoring Line 9 for now). The addition of `dropdown-menu-item`[s] relies on the input from the standard backend. Infeasible choices are indicated by red text. Note the use of negative literals with predicate `_any` to identify selections belonging to no stable model. The effect of this approach is illustrated in the second snapshot of Figure 5.

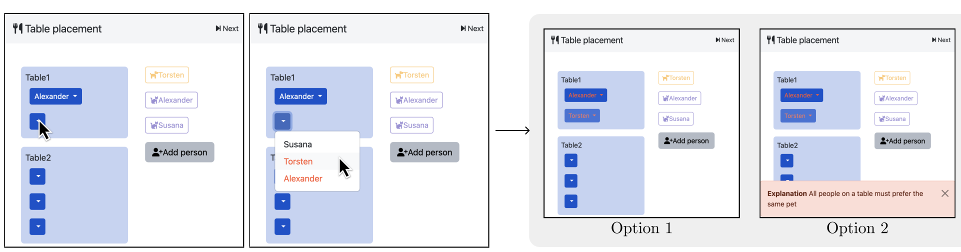


Figure 5: Example user interaction with the explanation extension.

Our first approach hinges on identifying user choices that prevent a solution. Here is where the *explanation backend* comes into play. Remember that we model user selections as assumptions. Modern

ASP solvers like *clasp* [3] or *wasp* [2] can pinpoint unsatisfiable sets of assumptions when solving an ASP program. We exploit this capability in *clinguin*'s explanation backend by including instances of the special predicate `_clinguin_mus/1` in the *domain-state*. The arguments of all such instances represent a minimal[9] set of assumptions that, when combined, lead to an unsatisfiable scenario.

Clinguin's `_clinguin_mus/1` predicate helps us pinpoint infeasible user choices. We leverage this predicate in Line 9 to highlight in red the `dropdown_menu`[s] of the assignments causing the issue (extending Lines 18 to 21 in Listing 3). Additionally, to only show the selection in menus associated with user choices, we modify Line 21 in Listing 3, and replace predicate `_all` with `_clinguin_assume` in the body of the rule:

```
21  attr(seat_dd(S), selected, P):- _clinguin_assume(assign(P,S),true).
```

This guarantees that the text in the dropdown menus reflects the user's selection, even if it leads to an unsatisfiable result. When an assumption causes an unsatisfiable scenario, the facts in the *domain-state* coming from the *domain-control* (including the stable model, `_any`, and `_all`) are retrieved from the last successful computation. Meanwhile, those indicating the state of unsatisfiability and selections (given by `_clinguin_mus/1` and `_clinguin_assume/2`) are effectively updated. Therefore, in this extension handling unsatisfiable scenarios, using predicate `_clinguin_assume` instead of `_all` ensures the selection is accurately displayed.

An interaction using this UI feature is shown in Figure 5, ending in Option 1. As before, we start by selecting Alexander for the first seat at Table 1. However, unlike Figure 2–4, the second seat remains unassigned due to the code modification in Listing 3. Clicking on this reveals both Torsten and Alexander highlighted in red. This signifies that neither option leads to a valid solution. Now, imagine a curious user selects Torsten despite the red color, aiming to understand why this choice is infeasible. As a result, the dropdown menus of the specific assumptions causing the unsatisfiability are highlighted in red. In this simple example, it might only indicate a conflict with Alexander's previous assignment. However, in more complex scenarios, only a subset of assumptions would typically be highlighted, pinpointing the exact source of the infeasibility.

The interaction leading to Option 1 in Figure 5 is obtained by the following command:

```
clinguin client-server --domain-files ins.lp enc.lp \
--ui-files ui-tables.lp ui-menu.lp ui-people.lp ui-explain.lp \
--backend ExplanationBackend
```

This command adds `ui-explain.lp` and the changed `ui-tables.lp` file to the UI files and moreover an additional parameter to indicate the use backend `ExplanationBackend`.

Our second part of the extension aims at providing user-friendly explanations in natural language when their choices lead to dead ends. For instance, in our running example, Alexander and Torsten cannot be placed at the same table because they have conflicting pet preferences. Ideally, the UI should display an error message that clearly explains this specific reason whenever both are assigned to the same table. The key to generating these explanations lies in their connection to the domain files. Since explanations are specific to the problem being modeled, we leverage the concept of integrity constraints defined in those files. By tying unsatisfiability to the violation of these constraints, we can precraft corresponding explanations that pinpoint the exact reason behind the dead end.

To this end, we use the binary predicate `cons/2` in Listing 2 to identify and trigger constraints. In essence, each integrity constraint is paired with a `cons/2` instance that acts as a translator, converting the technical constraint violation into an understandable explanation for the user. In order to have these

[9]The set is minimal in the sense that removing any of its assumptions leads to a satisfiable result.

atoms as part of our minimal set of assumptions, these backend extension performs a transformation of the domain files. This transformation will (internally) replace facts matching the predicate signature `cons/2` by a choice, providing the option to activate or deactivate the corresponding constraint when searching for unsatisfiable assumptions. Then, we treat each instance of `cons/2` as a true assumption, which keeps all original integrity constraints intact.

Now that we understand how `cons/2` instances link constraints to explanations, let us explore how these explanations are triggered within the system. Whenever a specific `cons/2` instance ends up in an unsatisfiable set of atoms, it indicates that the constraint must be active to trigger the inconsistency, which can be interpreted as a violation of the corresponding constraint. This violation triggers the explanation associated with that `cons/2` instance in the UI by opening a `message` box displaying the explanation. This is defined in Listing 7 for `cons/2` instances belonging to the current set of assumptions held in the server's *domain-state*.

```
elem(message_unsat(N), message, w):-_clinguin_mus(cons(N,M)).
attr(message_unsat(N), title, "Explanation"):-_clinguin_mus(cons(N,M)).
attr(message_unsat(N), message, M):-_clinguin_mus(cons(N,M)).
attr(message_unsat(N), type, error):-_clinguin_mus(cons(N,M)).
```

Listing 7: UI to handle complex explanations (`ui-explain-msg.lp`)

Finally, we can engage the interaction leading to Option 2 in Figure 5 as follows:

```
clinguin client-server --domain-files ins.lp enc.lp \
--ui-files ui-tables.lp ui-menu.lp ui-people.lp \
           ui-explain.lp ui-explain-msg.lp \
--backend ExplanationBackend --assumption-signature cons,2
```

We add `ui-explain-msg.lp` to the previous set of UI files, and indicate the spacial treatment of facts matching a predicate signature by using the command line option `--assumption-signature` in a generic way.

We now address the implementation details of this *clinguin* backend. The `ExplanationBackend` extends the available standard backend for *clingo*, viz `ClingoBackend`. In detail, it alters the setup to keep track of the internal representation of the assumptions, registers a new command line option `assumption-signature`, and adds (minimally) unsatisfiable sets of assumptions to the *domain-state*.

Listing 8 presents the complete Python implementation of the `ExplanationBackend` class. The `classmethod` in Lines 4 to 10 adds the `--assumption-signature` argument to the command line. Its value is then processed in the method `_init_command_line` to extract the predicate signatures, as shown in Lines 12 to 18. Of special interest is Line 18, which creates an object of the class `AssumptionTransformer`, defined by the Python library `clingexplaid` [10]. This library provides the necessary functionalities to transform the domain files and minimize the set of unsatisfiable assumptions. The transformer is then used in the `_load_file` method (Lines 24 to 26) to change how files are loaded into the *domain-state* in the standard backend. Specifically, it transforms the facts in these files into choices based on the given predicate signatures. This object is also used after grounding (Lines 28 to 32) to retrieve the list of assumptions from the predicate signature and store them. In Lines 21 to 22, these assumptions are included in the property `_assumption_list`, which is used as a parameter when solving. Finally, the domain constructor in Lines 39 to 44 returns a program that gets added to the *domain-state* by the method `_init_ds_constructors` in Lines 34 to 36. Line 40 adds a `#defined` statement for the predicate `_clinguin_mus/1` to avoid warnings. The code in Lines 41 to 43 is executed only when the output of the *domain-control* is unsatisfiable. In this case, it retrieves a minimal set of unsatisfiable

[10] `https://github.com/potassco/clingo-explaid`

assumptions using an object of the class `CoreComputer` from the same library, which is added to the *domain-state* (Line 43). The decorator `@cached_property` ensures efficient computation by reusing results unless the cache is invalidated.

4 Related work

The creation of problem-specific interfaces has been a long standing challenge for declarative methods. A Prolog-based method using XML was investigated in [15]. [14] focus on automatic user interface generation with model-based UIs. This is extended with contextual information and ASP in [16]. In the context of non-interactive ASP visualizations, recent advancements have led to *clingraph* [10]. The need for interactivity in ASP was addressed in previous studies [9], and later incorporated into *clingo*'s API with multi-shot capabilities [12], enabling continuous solving of logic programs that undergo frequent changes. Tools have also emerged to facilitate ASP program development, such as ASP Chef [1] for task pipelining, and various Integrated Development Environments [6, 4]. These tools align with advancements in related areas like argumentation [5]. In contrast to existing work, *clinguin* focuses on creating modern domain-specific interactive user interfaces in ASP.

5 Conclusion

We have presented *clinguin*, an easy yet expressive tool for creating user interfaces within ASP. This application relies on ASP features, like union and intersection of stable models, assumptions, and minimally unsatisfiable sets. A central idea is to reify these concepts and to keep them along with a stable model in focus as a set of atoms, which is then used by a UI encoding to generate a user interface and react to user events. Meanwhile, *clinguin* has become invaluable in our industrial applications since it greatly extends the rapid prototyping nature of ASP. *Clinguin* (version `2.0`) is freely available as open-source software at `https://github.com/potassco/clinguin`; its documentation is obtained at `https://clinguin.readthedocs.io`. The distribution contains several substantial use-cases, illustrating the full power of *clinguin*.

References

[1] M. Alviano, D. Cirimele & L. Rodriguez Reiners (2023): *Introducing ASP recipes and ASP Chef.* In J. Arias, S. Batsakis, W. Faber, G. Gupta, F. Pacenza, E. Papadakis, L. Robaldo, K. Rückschloß, E. Salazar, Z. Saribatur, I. Tachmazidis, F. Weitkämper & A. Wyner, editors: *Proceedings of the International Conference on Logic Programming Workshops (ICLP'23)*, *CEUR Workshop Proceedings* 3437, CEUR-WS.org. Available at `https://ceur-ws.org/Vol-3437`.

[2] M. Alviano, C. Dodaro, S. Fiorentino, A. Previti & F. Ricca (2023): *ASP and subset minimality: Enumeration, cautious reasoning and MUSes*. *Artificial Intelligence* 320, p. 103931, doi:10.1016/J.ARTINT.2023.103931.

[3] B. Andres, B. Kaufmann, O. Matheis & T. Schaub (2012): *Unsatisfiability-based optimization in clasp*. In A. Dovier & V. Santos Costa, editors: *Technical Communications of the Twenty-eighth International Conference on Logic Programming (ICLP'12)*, 17, Leibniz International Proceedings in Informatics (LIPIcs), pp. 212–221, doi:10.4230/LIPIcs.ICLP.2012.211.

[4] P. Busoniu, J. Oetsch, J. Pührer, P. Skocovsky & H. Tompits (2013): *SeaLion: An eclipse-based IDE for answer-set programming with advanced debugging support* Theory and Practice of Logic Programming 13, 4-5, pp. 657–673. doi:10.1017/S1471068413000410.

[5] R. Dachselt, S. A. Gaggl, M. Krötzsch, J Méndez, D. Rusovac & M. Yang (2022): *Nexas: A visual tool for navigating and exploring argumentation solution spaces*. In F. Toni, S. Polberg, R. Booth, M. Caminada & H. Kido, editors: *Proceedings of the Conference of Computational Models of Argument (COMMA'22)*, 353, IOS Press, pp. 116–127, doi:10.3233/FAIA220146. Available at `https://www.iospress.com/catalog/books/computational-models-of-argument-7`.

[6] O. Febbraro, K. Reale & F. Ricca (2011): *ASPIDE: Integrated Development Environment for Answer Set Programming*. In J. Delgrande & W. Faber, editors: *Proceedings of the Eleventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'11)*, *Lecture Notes in Artificial Intelligence* 6645, Springer-Verlag, pp. 317–330, doi:10.1007/978-3-642-20895-9_37.

[7] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub & S. Thiele (2015): *Potassco User Guide*, 2 edition. University of Potsdam. Available at `http://potassco.org`.

[8] M. Gebser, R. Kaminski, B. Kaufmann & T. Schaub (2019): *Multi-shot ASP solving with clingo*. *Theory and Practice of Logic Programming* 19(1), pp. 27–82, doi:10.1017/S1471068418000054.

[9] M. Gebser, P. Obermeier & T. Schaub (2015): *Interactive Answer Set Programming: Preliminary Report*. In S. Ellmauthaler & C. Schulz, editors: *Proceedings of the International Workshop on User-Oriented Logic Programming (IULP'15)*, doi:10.48550/arXiv.1511.01261. Available at `http://iulp2015.uni-leipzig.de/`.

[10] S. Hahn, O. Sabuncu, T. Schaub & T. Stolzmann (2022): *clingraph: ASP-based Visualization*. 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. 401–414, doi:10.1007/978-3-031-15707-3_31.

[11] T. Janhunen, R. Kaminski, M. Ostrowski, T. Schaub, S. Schellhorn & P. Wanko (2017): *Clingo goes Linear Constraints over Reals and Integers*. *Theory and Practice of Logic Programming* 17(5-6), pp. 872–888, doi:10.1017/S1471068417000242.

[12] 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.

[13] V. Lifschitz (2019): *Answer Set Programming*. Springer-Verlag, doi:10.1007/978-3-030-24658-7.

[14] E. Schlungbaum & T. Elwert (1996): *Automatic User Interface Generation from Declarative Models*. In J. Vanderdonckt, editor: *Computer-Aided Design of User Interfaces I, Proceedings of the Second International Workshop on Computer-Aided Design of User Interfaces*, Presses Universitaires de Namur, pp. 3–18, doi:10.13140/2.1.3102.3369.

[15] C. Schneiker & D. Seipel (2012): *Declarative Web Programming with PROLOG and XUL*. In: *Proceedings of the Twenty-sixth Workshop on Logic Programming (WLP'12)*.

[16] J. Zakraoui & W. Zagler (2011): *A logical approach to web user interface adaptation*. In A. Holzinger & K. Simonic, editors: *Information Quality in e-Health: 7th Conference of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society*, Springer, pp. 645–656, doi:10.1007/978-3-642-25364-5_46.

```
class ExplanationBackend(ClingoBackend):

    @classmethod
    def register_options(cls, parser):
        ClingoBackend.register_options(parser)

        parser.add_argument(
            "--assumption-signature",
            help="Signatures that will be considered as true assumptions",
            nargs="+")

    def _init_command_line(self):
        super()._init_command_line()
        self._assumption_sig = []
        for a in self._args.assumption_signature or []:
            name, arity = a.split(",")
            self._assumption_sig.append((name, int(arity)))
        self._assumption_transformer = AssumptionTransformer(self._assumption_sig)


    @property
    def _assumption_list(self):
        return self._assumptions.union(self._assumptions_from_signature)

    def _load_file(self, f):
        transformed_program = self._assumption_transformer.parse_files([f])
        self._ctl.add("base", [], transformed_program)

    def _ground(self, program="base", arguments=None):
        super()._ground(program, arguments)
        transformer_assumptions = self._assumption_transformer.get_assumption_symbols(
            self._ctl, self._ctl_arguments_list)
        self._assumptions_from_signature = [(a, True) for a in transformer_assumptions]


    def _init_ds_constructors(self):
        super()._init_ds_constructors()
        self._add_domain_state_constructor("_ds_mus")

    @cached_property
    def _ds_mus(self):
        prg = "#defined _clinguin_mus/1. "
        if self._unsat_core is not None:
            mus_core = CoreComputer(self._ctl, self._assumption_list).shrink()
            prg += " ".join([f"_clinguin_mus({str(s)})." for s, _ in mus_core])
        return prg + "\n"
```

Listing 8: Python implementation of ExplanationBackend

A Prolog Program for Bottom-up Evaluation

David S. Warren
Department of Computer Science
Stony Brook University
Stony Book, New York
warren@cs.stonybrook.edu

This short paper describes a simple and intuitive Prolog program, a metainterpreter, that computes the bottom up meaning of a simple positive Horn clause definition. It involves a simple transformation of the object program rules into metarules, which are then used by a metainterpreter to compute bottom up the model of the original program. The resulting algorithm is a form of semi-naive bottom-up evaluation. We discuss various reasons why this Prolog program is particularly interesting.

In particular, this is perhaps the only Prolog program for which I find the use of Prolog's `assert/1` to be intrinsic, easily understood, and the best, most perspicuous, way to program an algorithm. This short paper might be best characterized as a Prolog programming pearl.

1 Introduction

A positive Prolog program is an inductive definition of a set of relations over terms. There are two well-known basic ways to compute elements of relations that are defined in this way [2], [5]: top-down [4] and bottom-up [1].

Top-down evaluation starts with a query to a defined relation that asks for members of the defined relation that are instances of the query. It is query driven, since at each step a set of subqueries is posed that drive the search for an answer. Bottom-up evaluation starts with the facts, the data, and uses existing facts to iteratively infer new facts to ultimately compute all members of the defined relations.

Prolog uses SLD resolution, a top-down, query-driven evaluation strategy using highly optimized compilation techniques. Datalog systems use a bottom-up, data-driven evaluation strategy with engines optimized for that execution. Some problems are more efficiently solved top down and some bottom up. Thus there are some problems for which a Prolog programmer would like to use the Prolog built-in top-down strategy to perform bottom-up evaluation.

Several Prolog systems now support a hybrid evaluation strategy called *tabled evaluation* (e.g., [8],[7],[9],[3]), which allows those Prolog systems to use aspects of bottom-up evaluation within the Prolog top-down framework. Tabling normally can provide all the bottom-up processing needed by a Prolog programmer. But some Prolog systems do not have tabling, and tabling is not precisely bottom-up evaluation in any case.

Another use for bottom-up processing in Prolog is for teaching and debugging purposes. Students can learn more about what their Prolog definitions mean by exploring how they would evaluate bottom up. And experienced programmers may be able to find why Prolog does not compute an expected result by exploring the bottom-up computation.

For all these reasons a good, simple, customizable implementation of bottom-up evaluation within a Prolog environment is desirable. Prolog is well-known for the elegance of the metainterpreters that it supports, and a bottom-up evaluator requires a metainterpreter. But Prolog metainterpreters normally inherit a top-down strategy from the Prolog evaluator. A metainterpreter that interprets object programs bottom up is not so obvious.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 229–235, doi:10.4204/EPTCS.416.20

This short paper describes a metainterpreter written in (standard ISO) Prolog that evaluates pure, positive Prolog programs bottom up.

2 The Algorithm

The first step of the algorithm is to transform the object program into a new set of rules, rules that define a new binary predicate we call `implies/2`. The `implies` rules are generated from the object program as follows:

For each object rule:

$$H \text{ :- } B_1, B_2, \ldots, B_n.$$

generate n `implies/2` rules, one for each B_i. Specifically, for each B_i generate

$$\text{implies}(B_i, H) \text{ :- } B_1, B_2, \ldots, B_{i-1}, B_{i+1}, \ldots B_n.$$

And for each object fact of the form H. generate an `implies/2` fact of the form $\text{implies}(\text{true}, H)$.

The metainterpreter uses these generated `implies/2` rules to evaluate the original object program bottom up. The main metainterpreter predicate `bu(Schedule,SchedEnd)` takes a difference list that contains all the facts that have been derived but have not yet been used, along with the rules and other known facts, to derive new facts.

The Prolog code for the metainterpreter `bu/2` is:

```
bu([],[]).           % finished
bu([Fact|Sched],SchedEnd) :-
    nonvar(Fact),  % another derived fact to process
    findall(Hd,implies(Fact,Hd),Inferred),  % get newly inferred facts
    assert_and_sched_inferred(Inferred,SchedEnd,NewSchedEnd),
    bu(Sched,NewSchedEnd).
```

The first clause terminates when there are no more derived facts whose inferences are to be propagated. The second clause is the workhorse: it takes the next fact from the schedule, calls `implies/2` on that fact and collects together the heads of all object rules now satisfied using that fact, i.e. facts that can now be inferred. It then uses `assert_and_sched_inferred` to process all those facts: for each that is not already in the database, it adds it to the database and adds it to the end of the schedule. And iterates.

The supporting predicates are:

```
assert_and_sched_inferred([],Sched,Sched).
assert_and_sched_inferred([Fact|Inferred],Sched,SchedEnd) :-
    (subsuming_fact(Fact)
     -> assert_and_sched_inferred(Inferred,Sched,SchedEnd)
     ;  assert(Fact),
        Sched = [Fact|SchedTail],
        assert_and_sched_inferred(Inferred,SchedTail,SchedEnd)
    ).

subsuming_fact(Hd) :-
    term_variables(Hd,Variables),
    call(Hd),
    is_most_general_term(Variables).  % all still variables and distinct
```

The initial call to generate the least fixpoint is:

```
?- bu([true|End],End).
```

Consider the meaning of a generated rule for `implies/2` in `bu/2`. Atom `implies(A,H)` is true if there is a rule in the object program that has head `H` and a body literal `A` and all the other body literals of the rule are true in the database. I.e., `A` implies `H` if the remaining body literals are true. This is clearly a consequence of the original object program rule. These rules are applied starting with the empty database, finding facts implied by the database facts and these rules, updating the database with the newly inferred facts, and iterating until no new facts can be inferred.

3 Transitive Closure Example

Consider the Prolog rules for transitive closure of a simple graph:

```
tc(X,Y) :- edge(X,Y).
tc(X,Y) :- edge(X,Z), tc(Z,Y).

edge(a,b).
edge(b,c).
edge(c,b).
```

We note that since the graph in `edge/2` is a cyclic graph, Prolog will go into an infinite loop when asked the query `:- tc(A,B).` But bottom-up evaluation will terminate as we shall see.

The `implies/2` rules generated from these object rules and facts are:

```
implies(edge(X,Y),tc(X,Y)).

implies(edge(X,Z),tc(X,Y)) :- tc(Z,Y))
implies(tc(Z,Y),tc(X,Y)) :- edge(X,Z))

implies(true,edge(a,b)).
implies(true,edge(b,c)).
implies(true,edge(c,b)).
```

Each iteration of the main loop takes a fact from the schedule and produces a list of newly derived facts. that trace is:

```
true adds [edge(a,b),edge(b,c),edge(c,b)]
edge(a,b) adds [tc(a,b)]
edge(b,c) adds [tc(b,c)]
edge(c,b) adds [tc(c,b),tc(c,c)]
tc(a,b) adds []
tc(b,c) adds [tc(a,c)]
tc(c,b) adds [tc(b,b)]
tc(c,c), tc(a,c), and tc(b,b) each adds []
finished
```

This evaluator also handles unsafe programs, i.e., those with variables in answers. For example:

```
append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).
```

which generates the `implies/2` facts:

```
implies(true,append([],A,A).
implies(append(A,B,C),append([D|A],B,[D|C]).
```

and its trace:

```
true adds [append([],A,A)]
append([],A,A) adds [append([B],C,[B|C])]
append([B],C,[B|C]) adds [append([D,E],F,[D,E|F])]
append([D,E],F,[D,E|F]) adds [append([G,H,I],J,[G,H,I|J])]
and it continues...
```

Clearly if a Prolog program has infinitely many answers, its bottom-up evaluation will not terminate. But it can be useful to see the first several answers, as we've done here for `append/3`. We can modify our bottom-up evaluator to return nondeterministically each fact and the new facts it generates. This allows the user to see the facts as they are iteratively generated and either stop the generation or to ask for another. The following modification of `bu/2` is such a program:

```
bu([Fact|Sched],SchedEnd,Fact1,NewFacts) :-
    nonvar(Fact),  % done if a variable.
    findall(Hd,implies(Fact,Hd),Inferred),
    assert_and_sched_new(Inferred,SchedEnd,NewSchedEnd),
    (NewSchedEnd = [], SchedEnd \== [], % something added
     Fact1 = Fact, New = SchedEnd  % return this generator and generated set
     ;
     bu(Sched,NewSchedEnd,Fact1,NewFacts) % on to more, if asked
    ).
```

4 Why this Prolog Program Is Interesting

Why might this problem of bottom-up evaluation make for an interesting Prolog program?

1. It solves a basic problem in logic. This implementation in Prolog may provide evidence for a sometimes made claim that the Prolog programming language is a good assembler language, or implementation language, for logic-based problems.
2. It is a fundamental problem in the theory of logic programming.
3. It is a metainterpretation problem, for which Prolog is known for elegantly solving.

Prolog programs using `assert/1`, as this one does, do not have a declarative meaning and in general can be notoriously difficult to understand. This Prolog program is fairly easy to understand even though it uses `assert/1` in a fundamental way and does not therefore have a traditional declarative meaning. Nevertheless, it is perspicuous for several reasons:

1. The program is essentially deterministic. The persistence of asserted clauses over backtracking is one feature of `assert/1` that can make programs using it obscure. This program has essentially no backtracking.
2. The primary structure of this program is a single deterministic loop, clearly indicated by its tail recursive call. Each time around the loop adds a new set of facts to the database.

3. The nondeterminism, which is very convenient, is encapsulated by the `findall/3`) metacall. It simply collects the set of facts to be added to the database on the current iteration, which are nondeterministically generated by the call to the `implies/2` predicate.

4. The use of `assert/1` allows us to use Prolog's rule evaluation mechanism to determine what new atoms can be inferred on an iteration. The call to Prolog predicate `implies/1`, implicit in the `findall/3`, performs a metainterpretation step, at the object level.

5. The logical meaning of the `implies` clauses is clear and compelling.

These observations lead me to believe that this is an interesting Prolog program, even though it inherits no declarative meaning from Prolog's logical semantics. Its meaning comes from a combination of Prolog's procedural meaning (for the deterministic loop that updates the database) and its declarative meaning (for the call to `implies/2` and its accumulation with `findall/3`).

5 Complexity

On its face, this program seems efficient. It doesn't recompute in each step everything that was computed in the previous step as naive bottom-up evaluations does. I.e., it performs what is known as a semi-naive computation. Every call to `implies/2`, the workhorse of the computation, will use the standard first-argument indexing of all Prolog evaluators (but it may not be optimally indexed). And those calls use Prolog's built-in evaluation mechanism and so can avoid a level of metainterpretation. All this bodes well for good performance.

However, we can see a possible source of redundant computation. Consider an object program clause that has a large number of body atoms: H :- $B_1, B_2, ..., B_n$ for large n. There will be n `implies/2` clauses generated from it. Assume that the rule does indeed fire at some iteration. That will be after every B_i has been asserted into the database at some iteration. Say they are added in their left-to-right order, i.e., B_1 is added first, then B_2 at a later iteration, then B_3 at a sill later iteration, etc., until finally B_n is added and the `implies/2` rule for B_n causes the head to be added to the database. Notice that the first body atom B_1 will have been called $n-1$ times, once at each level that some B_i was added. And B_2 will be called $n-2$ times, once for each level after the first has been added. And so on. So there is $O(n^2)$ computation when only $O(n)$ is theoretically needed. (One could imagine removing a body literal when it becomes true. How to do this efficiently with the data structures used here is not so obvious.) Indeed the n is only the maximum number of literals in the body of any clause, which for normal programs is not very big. But it is still an unnecessary redundancy and for some programs may be very costly.

This happens only for programs with clauses with many body literals. If all program clauses have at most two body literals, then this redundancy does not occur. And any program can be transformed into one in which all rules have two or fewer body literals by a folding process that introduces intermediary predicates. E.g., H :- B_1, B_2, B_3 can be replaced by two rules H :- H', B_3 and H' :- B_1, B_2 where H' is new. This process reduces the number of literala of a rule by one, and can be iterated until we have at most two body literals for any rule. This program transformation was introduced in early Datalog systems, which are bottom up, for exactly this reason.

We noted that the `implies/2` predicate is by default indexed on its first argument, which is bound for every call. However, if the index is on only the main functor symbol, then the effective indexing is only on the predicate name of the newly added fact. Some Prolog's support deeper indexing, which would be helpful here.

6 Generating the implies/2 Rules

In this section we provide definitions for a couple of supporting functions: the generation of `implies/2` clauses from Prolog clauses, and the conversion of clauses with more than two body literals into a set of clauses with at most two by the introduction of intermediary predicates and folding.

The first predicate takes a list of Horn clauses and asserts the corresponding set of `implies/2` clauses. Other variations of this functionality might be more appropriate for integration into a larger system. For example, using a "term expansion" macro functionality of some Prologs might be effective.

```
assert_imply_rules([]).
assert_imply_rules([(Head:-Body)|Rules]) :-
    (do_all
     delete_one(Atom,Body,NBody),
     assert((implies(Atom,Head) :- NBody))
    ),
    assert_imply_rules(Rules).

delete_one(G1,G1,true) :- \+ G1 = (_,_).
delete_one(G1,(G1,G2),G2).
delete_one(G2,(G1,G2),G1) :- \+ G2 = (_,_).
delete_one(SG,(G1,G2),(G1,G3)) :-
    G2 = (_,_),
    delete_one(SG,G2,G3).
```

The predicate `delete_one/3` is like Prolog's `select/3` (sometimes called `delete/3`) for lists but works on comma-structures. The messy tests are because of their "default-y" representation [6] and to avoid an extraneous `true` goal at the end a clause.

The second functionality is the generation of folded clauses to provide better complexity for our bottom-up metainterpreter. For most applications envisioned this optimization may be unnecessary and in fact may complicate the understanding intended to be enhanced by exploring bottom-up evaluation. We also note that the choice of lierals to fold out and to keep and their order may affect the efficiency of the resulting rules. We give this simple definition here just for completeness.

The predicate `expand_rules` takes a list of clauses and returns an equivalent list of clauses each with two or fewer body literals:

```
expand_rules([],[]).
expand_rules([Rule|Rules],[NRule|ExpRules]) :-
    (gen_folded(Rule,NRule,TRule)
    -> expand_rules([TRule|Rules],ExpRules)
    ;  NRule = Rule,
       expand_rules(Rules,ExpRules)
    ).


gen_folded((Head :- Lit1,TBody),(Head:-Lit1,THead),(THead:-TBody)) :-
    TBody = (_,_),
    gensym('_$Tmp',Pred),
    Outer = (Lit1,Head),
```

```
excess_vars(Outer,TBody,[],Vs1),
excess_vars(TBody,Outer,Vs1,Vs2),
excess_vars((Outer,TBody),Vs1-Vs2,[],Vs),
THead =.. [Pred|Vs].
```

7 Conclusion

We have presented a metainterpreter written in Prolog to evaluate Horn clause programs bottom up. We find this program interesting for its simplicity and clarity. To understand it, one must understand part as a procedural computation and part as a declarative specification and integrate those understandings. To my mind it is unusual in that Prolog's `assert/1` operation is intrinsic and adds to the program's elegance.

References

[1] François Bancilhon (1985): *Naive Evaluation of Recursively Defined Relations*. In Michael L. Brodie & John Mylopoulos, editors: *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies, Book resulting from the Islamorada Workshop 1985 (Islamorada, FL, USA)*, Topics in Information Systems, Springer, pp. 165–178, doi:10.1007/978-1-4612-4980-1.

[2] Maarten H. van Emden & Robert A. Kowalski (1976): *The Semantics of Predicate Logic as a Programming Language*. *J. ACM* 23(4), pp. 733–742, doi:10.1145/321978.321991.

[3] Manuel Hermenegildo, Jose Morales, Manuel Carro, Pedro Lopez & Nik Swoboda (2024): *Ciao Documentation*. Available at https://ciao-lang.org/documentation.html.

[4] Robert A. Kowalski (1974): *Predicate Logic as Programming Language*. In Jack L. Rosenfeld, editor: *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974*, North-Holland, pp. 569–574.

[5] John W. Lloyd (1984): *Foundations of Logic Programming, 1st Edition*. Springer, doi:10.1007/978-3-642-96826-6.

[6] Richard A. O'Keefe (1990): *The Craft of Prolog*. MIT Press, Cambridge, MA, USA.

[7] Vitor Santos-Costa, Luis Damas, Rogerio Reis & Ruben Azevedo (2000): *YAP User Manual*. Available at https://www.dcc.fc.up.pt/~michel/yap.pdf.

[8] Theresa Swift, David S. Warren et al. (2024): *XSB User Manual*. Available at https://xsb.sourceforge.net/manual1/manual1.pdf.

[9] Jan Wielemaker (2024): *SWI User Manual*. Available at https://www.swi-prolog.org/pldoc/doc_for?object=manual.

Regular Typed Unification

João Barbosa Mário Florido
Departamento de Ciência de Computadores, Faculdade de Ciências, Universidade do Porto
Rua do Campo Alegre s/n, 4169–007 Porto, Portugal
LIACC - Laboratório de Inteligência Artificial e Ciência de Computadores

Vítor Santos Costa
Departamento de Ciência de Computadores, Faculdade de Ciências, Universidade do Porto
Rua do Campo Alegre s/n, 4169–007 Porto, Portugal
INESCTEC - Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência

Here we define a new unification algorithm for terms interpreted in semantic domains denoted by a subclass of regular types here called deterministic regular types. This reflects our intention not to handle the semantic universe as a homogeneous collection of values, but instead, to partition it in a way that is similar to data types in programming languages. We first define the new unification algorithm which is based on constraint generation and constraint solving, and then prove its main properties: termination, soundness, and completeness with respect to the semantics. Finally, we discuss how to apply this algorithm to a dynamically typed version of Prolog.

1 Introduction

In mathematical logic, a *term* denotes a mathematical object, and a theory of equality on the set of all terms formally defines which terms are considered equal. In logic programming terms are a syntactic representations of structured data such that in the typical case of first order languages, only syntactically identical terms are considered equal. Functions are thus uninterpreted or, computationally, functions build data terms, rather than operating on them.

When the domain of discourse contains elements of different kinds, it is useful to split the set of all terms (Universe) accordingly. To this end, a *type* (sometimes also called *sort*) is assigned to variables and constant symbols, and a declaration of the domain type and range type to each function symbol. A typed term $f(t_1,...,t_n)$ may then be composed from the subterms $t_1,...,t_n$ only if the i-th subterm's type matches the declared i-th domain type of f. Such a term is called *well-typed* and terms which are not well-typed are called *ill-typed*.

Previous approaches for types in logic programming use *regular types* as the type language. Some examples of that work are the works by Zobel [26, 7], Mishra [17], Yardeni [25], Fruhwirth *et al.* [9], Codish [4], Schrijvers *et al.* [21], Bruynooghe [20], Gallagher [10], Hermenegildo *et al.* [12], and Barbosa *et al.* [2], among others.

Data type definitions in programming languages impose constraints to the type language to allow decidable type checking. Namely, data types are recursive definitions where constructors are unique. Here we use such a subset of regular types that we shall call *deterministic regular types*, where each type constructor (here called *type function symbol*) is unique.

Type checking is most often done at compile-time, in order to ensure that program execution will not generate type errors at run-time. If typing is not possible, type errors may occur as the values of one or more arguments may not be in the expected domain. This work stems from the observation that logic programming systems will indeed output type errors in arguments of primitive predicates, but that there

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 236–252, doi:10.4204/EPTCS.416.21

is no way to check if unification of terms from different domains occurs. As a result, a program may generate several (unreported) errors and still succeed. The user may receive an unexpected answer, while having no insight on the existence of actual execution errors, thus making it difficult to detect and resolve program bugs. With this motivation in mind, we design a typed unification algorithm for typed first order theories, where types are described by deterministic regular types. This new unification algorithm may return three different results: a *most general unifier*, *failure* or *wrong*. This last value *wrong* is inspired by a similar notion used by Robin Milner to denote run-time type errors in functional programs [16] and, in our framework, it corresponds to the unification of terms that can never belong to the same semantic domains. A function now, may map integers to integers, integers to lists, floats to lists of integers, and, thus, the Herbrand universe is now divided into many different domains.

Example 1: Let $cons/2$ be the list constructor (in Prolog it would be denoted by ./2) with type $\forall\alpha.\alpha \times list(\alpha) \to list(\alpha)$, where $list(\alpha) = [\,] + [\alpha \mid list(\alpha)]$ (+ denotes type union). If we have terms $t_1 = cons(1,X)$ and $t_2 = cons(Y,2)$. These terms unify using first order (untyped) unification, but do not have a correct type, since the second argument of the list constructor must be a list. This is captured by the typed unification algorithm since it outputs *wrong*.

Contributions Our main contributions are: an extension to the semantics defined in [3], where equality takes into account the domains of the two terms in the left and the right hand side of an equation, being *wrong* when terms belong to disjoint domains; a *type system* for terms and the equality predicate which we prove to be sound with respect to the semantic typing relation; and a new unification algorithm, which given an equation between two terms returns a most general unifier for them and a *principal type*, if there is a solution, *false* if there is no solution but terms can belong to the same semantic domain and *wrong* otherwise. This three stage framework (first a notion of semantic typing, then a type system for terms and equations sound with respect to semantic typing and, finally, a unification algorithm sound and complete with respect to the type system) enables us to smoothly prove soundness and completeness of our unification algorithm, and it is inspired by the type theory in [16].

2 Term Syntax and Semantics

Here we define the language of terms, following [1, 14]. Given an infinite set of variables **VAR** and an infinite set of function symbols **FUNC**, a term is:

1. a variable (`X`, `Y`, `Xi`, ...);
2. a function symbol of arity 0 (`k`, `a`, `b`, `1`, ...), which we call a constant;
3. a function symbol of arity $n \geq 1$ (`f`, `g`, `h`, ...) applied to an n-tuple of terms.

We call terms that contain no variables *ground terms*, and terms that start with a function symbol with arity $n \geq 1$ *complex terms*.

Following the standard Herbrand interpretation of logic programs [1, 14], we assume that every ground term represents a tree and that all these trees are part of the universe of interpretation of the logic program.

We assume a particular partition of the universe into several domains. This interpretation groups sets of trees in the universe into domains, and includes some other domains that are not consisting of trees. We divide the universe **U** into domains as follows:

$$\mathbf{U} = \mathbf{Int} \cup \mathbf{Flt} \cup \mathbf{Str} \cup \mathbf{Atm} \cup \mathbf{List_1} \cup \ldots \cup \mathbf{List_n} \cup \mathbf{A_1} \cup \ldots \cup \mathbf{A_m} \cup \mathbf{Bool} \cup \mathbf{F} \cup \mathbf{Wrong},$$

where **Int** is the set of trees that represent integers (examples include `1` and `-10`, but also trees such as `1 + 4` and `2 * 5 - 1`), **Flt** is the set of trees that represent floating-point numbers, **Str** is the set of trees representing strings, **Atm** is the set of trees consisting of a single node, the root, that are not included in any other domain, $\mathbf{List_i}$ are sets of trees that represent lists, where each domain contains the trees that represent lists of elements of some other domain (i.e., we have a domain for lists of integers, lists of strings, lists of lists of integers, ...), $\mathbf{A_i}$ are the domains of trees whose root is a function symbol and the nodes of each tree are in the same domain as the corresponding nodes of every other tree (examples: $f(Int)$, $g(Int, Float)$, $h(g(Atom), h(Int))$, ...), **Bool** is the set with *true* and *false*, **F** is the set of functions, and **Wrong** is the set with the single value, *wrong*. We call *base domains* the domains **Int**, **Flt**, **Str**, and **Atm**.

One important note here is the value `[]`, corresponding to the empty list. We assume that this value belongs to every list domain, and that it is the only value that belongs to more than one domain in this partition.

We are using the domains for lists as an example of an interesting division of the universe that will later on correspond to inductively defined types. We could easily extend this partition by adding domains for other data types such as binary trees. We believe that any further domains for structured data can be extended easily following the approach we have for lists.

The semantics of a term is a tree in some domain, or *wrong*. The semantics depends on an interpretation I for the function symbols in the language, and a state Σ which associates variables to semantic values. We assume that the value returned by I is, for constants, a tree with just a root, and for function symbols of arity $n \geq 1$ a function in **F** that outputs a tree. Without loss of generality we assume that the only function symbol which does not have an Herbrand interpretation is the list constructor, thus for all function symbols `f` except the list constructor `cons`, the corresponding function in I is a function f that has signature $f : \forall \alpha_1, \ldots, \alpha_n . \alpha_1 \times \cdots \times \alpha_n \to f(\alpha_1, \ldots, \alpha_n)$, such that if any of the arguments the function is applied to is *wrong* then it outputs *wrong*, otherwise it outputs the tree with root f and children the trees it got as input. For the list constructor `cons` the function associated in I is *cons* with signature $cons : \forall \alpha . \alpha \times list(\alpha) \to list(\alpha)$ defined as:

$$cons(v_1, v_2) = \begin{cases} cons(v_1, v_2) & \text{if } v_1 \in D \land v_2 \in List(D) \\ wrong & \text{otherwise} \end{cases}$$

The predefined interpretation I is the one where every constant has the expected value, for instance the term `1` has as value the integer 1. One additional useful definition is the function *dom* that returns a set of domain of a value, a singleton set for all values but [ ].

We define the semantics of a term, represented by $[\![\ ]\!]_{I,\Sigma}$, in the following way:

- $[\![X]\!]_{I,\Sigma} = \Sigma(X)$
- $[\![k]\!]_{I,\Sigma} = I(k)$
- $[\![f(t_1, \ldots, t_n)]\!]_{I,\Sigma} = I(f)([\![t_1]\!]_{I,\Sigma}, \ldots, [\![t_n]\!]_{I,\Sigma})$

Note that, if a complex term contains the list constructor, the semantics of that term can be *wrong*. This is where the division into domains comes into play, since if we were considering an undivided Herbrand universe, then trivially all values are in the same domain so the application of a function could never generate an error. Informally, our approach supports the division of the universe implicit on the abstract data types definitions in the Prolog ISO standard (4.2) [8].

We assume only one predicate, equality, hereby represented by $=$. The semantics of equality is re-adjusted to take into account the value *wrong*. Equality is defined for terms in the same domain. So let function eq with signature $eq : \forall\alpha.\alpha \times \alpha \rightarrow Bool$ be defined as follows:

$$eq(v_1, v_2) = \begin{cases} true & \text{if } v_1 = v_2 \wedge dom(v_1) \cap dom(v_2) \neq \emptyset \wedge dom(v_1) \neq \{Wrong\} \\ false & \text{if } v_1 \neq v_2 \wedge dom(v_1) \cap dom(v_2) \neq \emptyset \wedge dom(v_1) \neq \{Wrong\} \\ wrong & \text{otherwise} \end{cases}$$

The semantics for the equality predicate is then as follows:

$$[\![t_1 = t_2]\!]_{I,\Sigma} = eq([\![t_1]\!]_{I,\Sigma}, [\![t_2]\!]_{I,\Sigma})$$

3 Types

Types are syntactic descriptions of semantic domains. The alphabet for the language of types includes an infinite set of type variables **TVar**, a finite set of base types **TBase**, an infinite set of type function symbols **TFunc**, an infinite set of type symbols **TSym**, parenthesis, and the comma. There is a one-to-one correspondence between **TFunc** and **FUNC**, which we assume is predefined. Then, we have the following grammar for types:

$$\begin{array}{lcl} all\_type & ::= & cons\_type \mid func\_type \\ cons\_type & ::= & type \mid type\_term \mid bool \\ func\_type & ::= & type_1 \times \cdots \times type_n \rightarrow type \mid type_1 \times \ldots \times type_n \rightarrow bool \\ type & ::= & tvar \mid tbase \mid tsymbol(type_1, \ldots, type_n) \\ type\_term & ::= & tconstant \mid tfunction(cons\_type_1, \ldots, cons\_type_n) \\ type\_def & ::= & tsymbol(tvar_1, \ldots, tvar_n) \longrightarrow type\_term_1 + \ldots + type\_term_m \end{array}$$

where $tvar \in$ **TVar**, $tbase \in$ **TBase**, $tconstant$ and $tfunction \in$ **TFunc**, and $tsymbol \in$ **TSym**. We call a type term that starts with a $tfunction$ a complex type term. We call *ground* to any type that does not contain a type variable.

Each type symbol is defined by a *type definition*. A *well-formed* type definition has all type variables that occur as parameters on the left-hand side of the definition be distinct and occurring somewhere on the right-hand side, and all type variables that occur on the right-hand side be a parameter on the left-hand side. The sum $\tau_1 + \ldots + \tau_n$ is a *union type*, describing values that have one of the type terms $\tau_1, \ldots, \tau_n$, called the *summands*. The '+' is an idempotent, commutative, and associative operation.

A set of type definitions D is called *deterministic* if it is well-formed and any type function symbol occurs at most once in D. In [7], the authors introduce the concept of *deterministic type definition*. Our definition is stricter than this previous one by disallowing base types and variables as summands in type definitions and disallowing more than one occurrence of any function symbol in the whole set of type definitions.

Deterministic type definitions include tuple-distributive types [26, 17] and correspond to the widely used algebraic data types in programming languages. From now on we assume that type definitions are deterministic.

A *type scheme* σ is an expression of the form $\forall\alpha_1, \ldots, \alpha_n.\tau$, where τ is a *type* or a $func\_type$ and $\alpha_1, \ldots, \alpha_n$ are type variables which will be called the *generic variables* of σ. If τ has no variables, then it is itself a type scheme. Note that types form a subclass of type schemes. We will abbreviate type schemes to $\forall\vec{\alpha}.\tau$, where $\vec{\alpha}$ denotes a sequence of several type variables α_i. Type schemes represent parametric polymorphic types [6].

4 Semantics

Each instance of a *type* is associated with a domain. A base type is associated with a base domain, and each instance of a type of the form $tsymbol(type_1,\ldots,type_n)$ is associated with a domain. We include a type symbol *list* that is associated with the domains for lists. We assume that the definition for the type symbol *list* is: $list(\alpha) \longrightarrow [\,] + cons(\alpha, list(\alpha))$. We could include further type symbols that were defined by inductive definitions, besides lists, and the rest of this paper could be easily extended to include different inductively defined types, but we keep *list* as the only one for the sake of simplicity.

A valuation ψ maps each type variable to a ground type. Given a valuation ψ, we define the semantics of a *type* and the type *bool* as follows:

$\mathbf{T}[\![\alpha]\!]_\psi = \mathbf{T}[\![\psi(\alpha)]\!]_\psi$
$\mathbf{T}[\![int]\!]_\psi = \mathbf{Int}$
$\mathbf{T}[\![bool]\!]_\psi = \mathbf{Bool}$
$\mathbf{T}[\![float]\!]_\psi = \mathbf{Flt}$
$\mathbf{T}[\![string]\!]_\psi = \mathbf{Str}$
$\mathbf{T}[\![atom]\!]_\psi = \mathbf{Atm}$
$\mathbf{T}[\![list(\alpha)]\!]_\psi = \mathbf{T}[\![list(\psi(\alpha))]\!]_\psi$
$\mathbf{T}[\![list(int)]\!]_\psi = List_i$, where $List_i$ is the domain for lists of integer. Similarly for any other ground instance of $list(\alpha)$ and the corresponding domain $List_j$.

The semantics of a *type_term* is:
$\mathbf{T}[\![k]\!]_\psi = \{k\}$
$\mathbf{T}[\![f(\tau_1,\ldots,\tau_n)]\!]_\psi = \{f(v_1,\ldots,v_n) \mid v_i \in \mathbf{T}[\![\tau_i]\!]_\psi\}$

The semantics of a *union type* is:
$\mathbf{T}[\![\tau_1 + \ldots + \tau_n]\!]_\psi = \mathbf{T}[\![\tau_1]\!]_\psi \cup \ldots \cup \mathbf{T}[\![\tau_n]\!]_\psi$

The semantics of a *func_type* is:
$\mathbf{T}[\![\tau_1 \times \ldots \times \tau_n \to \tau]\!]_\psi = \{f \mid f \in \mathbf{F} \wedge (v_1 \in \mathbf{T}[\![\tau_1]\!]_\psi \wedge \ldots \wedge v_n \in \mathbf{T}[\![\tau_n]\!]_\psi \implies f(v_1,\ldots,v_n) \in \mathbf{T}[\![\tau]\!]_\psi)\}$

The semantics of a *type scheme* is:
$\mathbf{T}[\![\forall\vec{\alpha}.\tau]\!]_\psi = \bigcap_{\forall\vec{\sigma}} \mathbf{T}[\![\tau[\vec{\alpha} \mapsto \vec{\sigma}]]\!]_\psi$, where $\vec{\sigma}$ is a sequence of *types* of the same size as $\vec{\alpha}$.

Note that the semantics of a (ground) *type_term* may be a domain, as in the case of $f(int, float)$, or the subset of a domain, as in the case of $cons(int,[\,])$, or even a subset of several domains, as in the case of $[\,]$. This includes the domain **Wrong**, as in the case of $cons(int,int)$. All instances of complex type terms whose *tfunction* is not the list constructor are associated with a domain for trees.

Also note that we assume a function, given by I, for the interpretation of function symbols, thus functions have type signatures: the type of a function symbol f of arity n is interpreted as a function which builds a tree of root f, with the type scheme $\forall\alpha_1,\ldots,\alpha_n.\alpha_1 \times \cdots \times \alpha_n \to f(\alpha_1,\ldots,\alpha_n)$. The semantics for this type scheme is the intersection of the semantics for all instances of the functional type, which is a subset of $\mathbf{F}$ consisting of all functions that have such type. So it consists of all the functions that can have any tuple of n elements as input and output a tree whose root is f and the children nodes are the input elements.

4.1 Semantic Typing

We now define what it means for a term to semantically have a type, denoted by $t : \tau$. If the term and the type are both ground, given an interpretation I, we just check whether the semantics of the term belongs to the domain corresponding to the semantics of the type. So, for ground terms and types:

$$t : \tau \implies \forall \Sigma . \forall \psi . [\![t]\!]_{I,\Sigma} \in \mathbf{T}[\![\tau]\!]_{\psi}$$

However, both terms and types can be non-ground in general and, without extra information, we cannot know what is the correct type for a variable. To deal with variables we introduce the concept of a context Γ, defined as a set of typings of the form $X : \tau$ for variables. Given a context we define the *semantic typing* relation, denoted by $\models$, as:

$$\Gamma \models_I t : \tau \implies \forall \Sigma . \forall \psi . (\forall (X : \tau\prime) \in \Gamma . [\![X]\!]_{I,\Sigma} \in \mathbf{T}[\![\tau\prime]\!]_{\psi} \implies [\![t]\!]_{I,\Sigma} \in \mathbf{T}[\![\tau]\!]_{\psi})$$

We call the *generic context* to the context that contains $X_i : \alpha_i$, for all variables, i.e., all term variables are typed by a type variable, and each type variable is associated with a particular variable. Note that throughout the paper we will use the symbol $\models$ overloaded for other semantics relations.

Example 2: Let $\Gamma = \{\mathtt{X} : \alpha, \mathtt{Y} : list(\alpha)\}$ and the type signature for `cons` in I be $\{\mathtt{cons} : \forall \alpha . \alpha \times list(\alpha) \to list(\alpha)\}$.

$$\Gamma \models_I \mathtt{cons(X,Y)} : list(\alpha)$$

Suppose we have a state Σ and a valuation ψ such that $[\![\mathtt{X}]\!]_{I,\Sigma} \in \mathbf{T}[\![\alpha]\!]_{\psi}$ and $[\![\mathtt{Y}]\!]_{I,\Sigma} \in \mathbf{T}[\![list(\alpha)]\!]_{\psi}$, then $[\![\mathtt{cons(X,Y)}]\!]_{I,\Sigma} \in \mathbf{T}[\![list(\alpha)]\!]_{\psi}$. Since $[\![\mathtt{X}]\!]_{I,\Sigma} = \Sigma(X) \in \mathbf{T}[\![\psi(\alpha)]\!]_{\psi}$ and $[\![\mathtt{Y}]\!]_{I,\Sigma} = \Sigma(Y) \in \mathbf{T}[\![list(\alpha)]\!]_{\psi} = \mathbf{T}[\![list(\psi(\alpha))]\!]_{\psi}$, by the semantics of `cons`, we have $cons(\Sigma(X), \Sigma(Y))$, which is not *wrong* from the domains of the respective values, and because the output is in the correct domain.
However, note that for $\Gamma\prime = \{\mathtt{X} : \alpha, \mathtt{Y} : \beta\}$, the same would not be true, since for $\Sigma = [X \mapsto 1, Y \mapsto 2]$ and $\psi = [\alpha \mapsto int, \beta \mapsto int]$, the left-hand side of the implication is true, but $cons(1,2) \notin \mathbf{T}[\![list(int)]\!]_{\psi}$.

5 Syntactic Typing

Syntactic typing is defined by a type system. A context Γ and a set of type assumptions for constants and function symbols Δ are needed to derive a type assignment and one writes $\Gamma, \Delta \vdash t : \tau$ (pronounce this as Γ and Δ yield t in τ). Assumptions in Δ are of the form $k : \forall \vec{\alpha} . \tau$, for constants, and $f : \forall \vec{\alpha} . \tau_1 \times \cdots \times \tau_n \to \tau$, for function symbols, where the generic variables $\vec{\alpha}$ of these type schemes are exactly the type variables that occur in τ and $\tau_1 \times \cdots \times \tau_n \to \tau$, respectively. A statement $t : \tau$ is *derivable* from contexts Γ and Δ, notation $\Gamma, \Delta \vdash t : \tau$, if it can be produced by the rules in Figure 1. If we have a derivation in the type system, then we say that t has type τ in contexts Γ and Δ.

We must guarantee that Δ is in agreement with I. For this, we have the following relation: $I \models \Delta$, is defined as $\forall (k : \tau) \in \Delta . dom(I(k)) = \{\tau\} \wedge \forall (f : \tau_1 \times \ldots \times \tau_n \to \tau) . I(f) : \tau_1 \times \ldots \times \tau_n \to \tau$.

Example 3: Let $\Gamma = \{X : int,\ Y : list(int)\}$, $\Delta = \{1 : int,\ nil : \forall \gamma . list(\gamma),\ cons : \forall \beta . \beta \times list(\beta) \to list(\beta)\}$, and $\Lambda = (cons : \forall \beta . \beta \times list(\beta) \to list(\beta)) \in \Delta$ (we use this Λ just to improve presentation). Then the following type derivation holds using the type rules:

$$\text{VAR}\ \frac{(X:\tau)\in\Gamma}{\Gamma,\Delta\vdash X:\tau} \qquad\qquad \text{CST}\ \frac{(k:\forall\vec{\alpha}.\tau)\in\Delta}{\Gamma,\Delta\vdash k:\tau[\vec{\alpha}\mapsto\vec{\sigma}]}$$

$$\text{CPL}\ \frac{\begin{array}{c}(f:\forall\vec{\alpha}.\tau_1\times\cdots\times\tau_n\to\tau)\in\Delta\\ \Gamma,\Delta\vdash t_1:\tau_1[\vec{\alpha}\mapsto\vec{\sigma}]\ \ \ldots\ \ \Gamma,\Delta\vdash t_n:\tau_n[\vec{\alpha}\mapsto\vec{\sigma}]\end{array}}{\Gamma,\Delta\vdash f(t_1,\ldots,t_n):\tau[\vec{\alpha}\mapsto\vec{\sigma}]} \qquad \text{EQU}\ \frac{\Gamma,\Delta\vdash t_1:\tau \quad \Gamma,\Delta\vdash t_2:\tau}{\Gamma,\Delta\vdash t_1=t_2:bool}$$

Figure 1: Type System

$$\dfrac{\dfrac{\dfrac{(X:int)\in\Gamma}{\Gamma,\Delta\vdash X:int}\quad\dfrac{([\,]:\forall\gamma.list(\gamma))\in\Delta}{\Gamma,\Delta\vdash[\,]:list(int)^{(2)}}\ \Lambda}{\Gamma,\Delta\vdash cons(X,[\,]):list(int)^{(1)}}\quad\dfrac{\dfrac{(1:int)\in\Delta}{\Gamma,\Delta\vdash 1:int}\quad\dfrac{(Y:list(int))\in\Gamma}{\Gamma,\Delta\vdash Y:list(int)}\ \Lambda}{\Gamma,\Delta\vdash cons(1,Y):list(int)^{(2)}}}{\Gamma,\Delta\vdash cons(X,[\,])=cons(1,Y):bool}$$

Note that in $^{(1)}$ we used $list(\beta)[\beta\mapsto int]$ and in $^{(2)}$ we used $list(\gamma)[\gamma\mapsto int]$. Also note that if $X:\alpha$ instead of $X:int$ was in Γ, we could not have a derivation.

We now prove that the rules for syntactic typing are sound, that is, if the set Δ is in agreement with I, then any type derivation is semantically correct.

Theorem 1 - Soundness of Syntactic Typing: If $\Gamma,\Delta\vdash t:\tau$ and $I\models\Delta$, then $\Gamma\models_I t:\tau$.

Proof: We will prove this by induction on the derivation.

- If the term t is a variable X, then the derivation consists of a single application of axiom *VAR*. Clearly, it is also true that $\Gamma\vdash_I X:\tau$, where $(X:\tau)\in\Gamma$, since any Σ that gives values to X and ψ that gives values to τ, such that $[\![X]\!]_{I,\Sigma}\in\mathbf{T}[\![\tau]\!]_\psi$ will do so in the context and in the term itself simultaneously, so $\Gamma\vdash_I X:\tau$.
- If the term t is a constant k, then the derivation consists of a single application of axiom *CST*. Since $I\models\Delta$, $dom(I(k))=\{\forall\vec{\alpha}.\tau\}$, where $(k:\forall\vec{\alpha}.\tau)\in\Delta$, then $k\in\mathbf{T}[\![\forall\vec{\alpha}.\tau]\!]_\psi$, for any ψ. But since $\mathbf{T}[\![\forall\vec{\alpha}.\tau]\!]_\psi=\bigcap_{\forall\vec{\sigma}}\mathbf{T}[\![\tau[\vec{\alpha}\mapsto\vec{\sigma}]]\!]_\psi$, then $k\in\mathbf{T}[\![\tau[\vec{\alpha}\mapsto\vec{\sigma}]]\!]_\psi$. So for any Σ, the right-hand side of the implication is always true, so $\Gamma\vdash_I k:\tau[\vec{\alpha}\mapsto\vec{\sigma}]$.
- If the term t is a complex term $f(t_1,\ldots,t_n)$, then we can assume, by induction hypothesis, that $\Gamma\vdash_I t_i:\tau_i[\vec{\alpha}\mapsto\vec{\sigma}]$, for all $i=1,\ldots,n$. Since $I\models\Delta$, $I(f):\forall\vec{\alpha}.\tau_1\times\cdots\times\tau_n\to\tau$, then $f\in\mathbf{T}[\![\forall\vec{\alpha}.\tau_1\times\ldots\times\tau_n\to\tau]\!]_\psi$, for all ψ, so $f\in\mathbf{T}[\![(\tau_1\times\ldots\times\tau_n\to\tau)[\vec{\alpha}\mapsto\vec{\sigma}]]\!]_\psi$. Therefore, we know that, if $v_i\in\mathbf{T}[\![\tau_i[\vec{\alpha}\mapsto\vec{\sigma}]]\!]_\psi$ then $f(v_1,\ldots,v_n)\in\mathbf{T}[\![\tau[\vec{\alpha}\mapsto\vec{\sigma}]]\!]_\psi$. For any Σ and ψ such that $\forall(X:\tau\prime)\in\Gamma.[\![X]\!]_{I,\Sigma}\in\mathbf{T}[\![\tau\prime]\!]_\psi$, by the induction hypothesis $[\![t_i]\!]_{I,\Sigma}\in\mathbf{T}[\![\tau_i[\vec{\alpha}\mapsto\vec{\sigma}]]\!]_\psi$. Therefore, for the same Σ and ψ, we know that $[\![f(t_1,\ldots,t_n)]\!]_{I,\Sigma}\in\mathbf{T}[\![\tau[\vec{\alpha}\mapsto\vec{\sigma}]]\!]_\psi$, so $\Gamma\vdash_I f(t_1,\ldots,t_n):\tau[\vec{\alpha}\mapsto\vec{\sigma}]$.
- If we have an equality of two terms $t_1=t_2$, we can assume, by induction hypothesis, that $\Gamma\vdash_I t_1:\tau$ and $\Gamma\vdash_I t_2:\tau$. Therefore we know that for any Σ and ψ such that $\forall(X:\tau\prime)\in\Gamma.[\![X]\!]_{I,\Sigma}\in\mathbf{T}[\![\tau\prime]\!]_\psi$,

we have $[\![t_1]\!]_{I,\Sigma} \in \mathbf{T}[\![\tau]\!]_\psi$ and $[\![t_2]\!]_{I,\Sigma} \in \mathbf{T}[\![\tau]\!]_\psi$. So for these Σ and ψ, we have $[\![t_1 = t_2]\!]_\Sigma \in [\![bool]\!]_\psi$. Therefore, $\Gamma \vdash_I t_1 = t_2 : bool$.

□

Given a term t is there a typing representing all possible typings of t? In order to answer this question we introduce the notion of *principal typing*, [13], as appropriate to our system.

Definition 1: A *principal typing* is a pair (Γ, τ), such that $\Gamma, \Delta \vdash t : \tau$ and for every other pair $(\Gamma\prime, \tau\prime)$ such that $\Gamma\prime, \Delta \vdash t : \tau\prime$, there is a type substitution μ such that $\mu(\Gamma) = \Gamma\prime$ and $\mu(\tau) = \tau\prime$.

Note that even though, initially, it might seem possible that the context in a principal typing will always be a generic context, for some cases that is not the case.

Example 4: Let $t =$`cons(X,Y)`. A principal typing for t is $(\{X : \alpha, Y : list(\alpha)\}, list(\alpha))$. Note that any renaming of type variable α defines another principal typing, because principal typings are unique up to renaming of type variables. Also note that the type for Y cannot be a type variable, thus, in this example, the context is not generic.

6 Constraints

To check implicit types during unification, we must deduce types that are not present in equality equations. To represent this problem in a broader context, we introduce the notion of type constraint which we add to the usual term unification problem.

We define equality constraints between terms $t_1 = t_2$, and equality constraints between types $\tau_1 \doteq \tau_2$. We are here using the same symbol for equality constraints and the equality predicate. We argue that the uses are clear from the context.

We say that a set of equality constraints is in *normal form* if all constraints are of the form $X_i = t_i$, for some term t_i, and there is no other occurrence of any X_i anywhere else in the set. A set of equality constraints in normal form can be interpreted as a substitution, where every constraint of the form $X_i = t_i$ is interpreted as $[X_i \mapsto t_i]$.

A set of type equality constraints is in normal form if all constraints are of the form $\alpha_i \doteq \tau_i$, for some type τ_i, and there is no other occurrence of any α_i anywhere else in the set. A set of type equality constraints in normal form can be interpreted as a type substitution, where every constraint of the form $\alpha_i \doteq \tau_i$ is interpreted as $[\alpha_i \mapsto \tau_i]$.

Definition 2: A substitution θ (or type substitution μ) is called a *unifier* for terms t_1 and t_2 (or types τ_1 and τ_2), iff $\theta(t_1) = \theta(t_2)$ (or $\mu(\tau_1) = \mu(\tau_2)$). Terms t_1 and t_2 (or types τ_1 and τ_2) are *unifiable* iff there exists a unifier for them.

Our constraints are supposed to represent equality, either of terms or types. However, in the semantics, we need states and valuations to interpret non-ground terms and types, respectively. Therefore, we need a way to interpret constraints semantically, so we define the following.

Definition 3: Let c be a constraint, Σ a state, and ψ a valuation. We say that Σ and ψ model c, and represent it by $\Sigma, \psi \models c$ if:

- c is an equality constraint of the form $t_1 = t_2$, then $[\![t_1]\!]_{I,\Sigma} = [\![t_2]\!]_{I,\Sigma}$;
- c is a type equality constraint of the form $\tau_1 \doteq \tau_2$, then $\mathbf{T}[\![\tau_1]\!]_{\psi} = \mathbf{T}[\![\tau_2]\!]_{\psi}$;

We can easily extend this definition for sets of constraints.

Definition 4: Let C be a set of equality constraints and S be a set of type equality constraints. We say that a state Σ and a valuation ψ model the pair (C,T), and represent it by $\Sigma,\psi \models C,T$ iff Σ and ψ model all constraints in both sets.

We now provide an auxiliary definition that relates substitutions and states and use this definition to extend our notion of constraint modelling.

Definition 5: We say that a state Σ follows a substitution θ and represent it by $\Sigma \sim \theta$ iff for any term t, $[\![t]\!]_{I,\Sigma} = v$ and $[\![\theta(t)]\!]_{I,\Sigma} = v$. Similarly, a valuation ψ follows a substitution for types μ ($\psi \sim \mu$) iff for any type τ, $\mathbf{T}[\![\tau]\!]_{\psi} = \mathbf{T}[\![\mu(\tau)]\!]_{\psi}$.

Definition 6: Let C be a set of equality constraints and S be a set of type equality constraints. We say that a substitution θ and a type substitution μ model the pair (C,T), and represent it by $\theta,\mu \models C,T$, iff for every state Σ and valuation ψ we have that $\Sigma \sim \theta \wedge \psi \sim \mu \implies \Sigma,\psi \models C,T$.

7 Typed Unification Algorithm

The typed unification algorithm performs unification for terms and types. The intuition is that if the types do not unify, then there is a type error. We will prove this condition in the next section. We follow the approach of [24]: generate constraints for typeability and solve them.

7.1 Constraint Generation

$$\text{GVAR}\ \frac{(X:\alpha)\in\Gamma}{\Gamma,\Delta\vdash X:\alpha \mid \emptyset \mid \emptyset} \qquad \text{GCST}\ \frac{(k:\forall\vec{\alpha}.\tau)\in\Delta}{\Gamma,\Delta\vdash k:\tau[\vec{\alpha}\mapsto\vec{\beta}] \mid \emptyset \mid \emptyset}$$

$$\text{GCPL}\ \frac{(f:\forall\vec{\alpha}.\tau_1\times\cdots\times\tau_n\to\tau)\in\Delta \qquad \Gamma,\Delta\vdash t_1:\tau\prime_1 \mid \emptyset \mid T_1 \ \ \ldots\ \ \Gamma,\Delta\vdash t_n:\tau\prime_n \mid \emptyset \mid T_n}{\Gamma,\Delta\vdash f(t_1,\ldots,t_n):\tau[\vec{\alpha}\mapsto\vec{\beta}] \mid \emptyset \mid T_1\cup\cdots\cup T_n\cup\{\tau\prime_1\doteq\tau_1[\vec{\alpha}\mapsto\vec{\beta}],\ldots,\tau\prime_n\doteq\tau_n[\vec{\alpha}\mapsto\vec{\beta}]\}}$$

$$\text{GEQU}\ \frac{\Gamma,\Delta\vdash t_1:\tau_1 \mid C_1 \mid T_1 \qquad \Gamma,\Delta\vdash t_2:\tau_2 \mid C_2 \mid T_2}{\Gamma,\Delta\vdash t_1=t_2:\mathit{bool} \mid \{t_1=t_2\} \mid T_1\cup T_2\cup\{\tau_1\doteq\tau_2\}}$$

Figure 2: Constraint Typing Judgment

Guided by the definition of our type system we now define a *constraint typing judgment*, which indicates what constraints must hold for a particular type term-and-context pair to be typeable.

Let Γ be a generic context, and Δ a set of type assumptions for constants and function symbols. We use the following rules to generate constraints for the unification of two terms t_1 and t_2. The generated constraints will be the pair (C,T) in $\Gamma,\Delta \vdash t_1 = t_2 : bool \mid C \mid T$. In the rules in Figure 2, $\vec{\beta}$ represents a sequence of fresh type variables of the same size as $\vec{\alpha}$ in the corresponding case.

Example 5: Let Γ be a generic context (we will denote the type variable associated with each variable X by α_X), $\Delta = \{1 : int,\ [\,] : \forall\alpha.list(\alpha),\ cons : \forall\beta.\beta \times list(\beta) \to list(\beta)\}$, $C = \{cons(X,[\,]) = cons(1,Y)\}$, and $\Lambda = (cons : \forall\beta.\beta \times list(\beta) \to list(\beta)) \in \Delta$. The following constraint type judgements hold:

$$\dfrac{\dfrac{(X : \alpha_X) \in \Gamma}{\Gamma,\Delta \vdash X : \alpha_X \mid \emptyset \mid \emptyset} \quad \dfrac{([\,] : \forall\alpha.list(\alpha)) \in \Delta}{\Gamma,\Delta \vdash [\,] : list(\gamma) \mid \emptyset \mid \emptyset} \;\; \Lambda}{\Gamma,\Delta \vdash cons(X,[\,]) : list(\nu) \mid \emptyset \mid \{\alpha_X = \nu,\ list(\gamma) = list(\nu)\}(= T_1)}$$

$$\dfrac{\dfrac{(1 : int) \in \Delta}{\Gamma,\Delta \vdash 1 : int \mid \emptyset \mid \emptyset} \quad \dfrac{(Y : \alpha_Y) \in \Gamma}{\Gamma,\Delta \vdash Y : \alpha_Y \mid \emptyset \mid \emptyset} \;\; \Lambda}{\Gamma,\Delta \vdash cons(1,Y) : list(\eta) \mid \emptyset \mid \{int = \eta,\ \alpha_Y = list(\eta)\}(= T_2)}$$

$$\dfrac{\Gamma,\Delta \vdash cons(X,[\,]) : list(\eta) \mid \emptyset \mid T_1 \quad \Gamma,\Delta \vdash cons(1,Y) : list(\nu) \mid \emptyset \mid T_2}{\Gamma,\Delta \vdash cons(X,[\,]) = cons(1,Y) : bool \mid C \mid T_1 \cup T_2 \cup \{list(\nu) = list(\eta)\}}$$

We will now prove that constraint generation is sound, i.e., if we generate constraints any model for them applied to Γ and type τ is derivable in the type system.

Theorem 2 - Soundness of the Constraint Generation: If $\Gamma,\Delta \vdash t : \tau \mid C \mid T$ and $\mu \models T$, then $\mu(\Gamma),\Delta \vdash t : \mu(\tau)$ is derivable in the type system.

Proof: We will prove this theorem by induction on the derivation.

- If t is a variable X, then we have $\Gamma,\Delta \vdash X : \alpha \mid \emptyset \mid \emptyset$. Any type substitution μ is such that $\mu \models \emptyset$. And, for any μ, since $\mu(\alpha)$ will be the same in Γ and in the consequent of the rule, $\mu(\Gamma),\Delta \vdash X : \mu(\alpha)$ is derivable in the type system by a single application of rule VAR.
- If t is a constant k, then we have $\Gamma,\Delta \vdash k : \tau[\vec{\alpha} \mapsto \vec{\beta}] \mid \emptyset \mid \emptyset$, where $(k : \forall\vec{\alpha}.\tau) \in \Delta$. Any type substitution μ is such that $\mu \models \emptyset$. Then, for any such μ we can have the derivation in the syntactic system using a single application of rule CST, using $\mu(\tau[\vec{\alpha} \mapsto \vec{\beta}]) = \tau[\vec{\alpha} \mapsto \mu(\vec{\beta})]$.
- If t is a complex term $f(t_1,\ldots,t_n)$, then we have $\Gamma,\Delta \vdash f(t_1,\ldots,t_n) : \tau[\vec{\alpha} \mapsto \vec{\beta}] \mid \emptyset \mid T_1 \cup \ldots \cup T_n \cup \{\tau\prime_1 \doteq \tau_1[\vec{\alpha} \mapsto \vec{\beta}],\ldots,\tau\prime_n \doteq \tau_n[\vec{\alpha} \mapsto \vec{\beta}]\}$, given $\Gamma,\Delta \vdash t_i : \tau_i\prime \mid \emptyset \mid T_i$, for $i = 1,\ldots,n$. We also know that $\mu \models T_1 \cup \cdots \cup T_n \cup \{\tau\prime_1 \doteq \tau_1[\vec{\alpha} \mapsto \vec{\beta}],\ldots,\tau\prime_n \doteq \tau_n[\vec{\alpha} \mapsto \vec{\beta}]\}$, and any such μ is such that $\mu \models T_i$ and $\mu \models \tau\prime_i \doteq \tau_i[\vec{\alpha} \mapsto \vec{\beta}]$, for each $i = 1,\ldots,n$. By the induction hypothesis, we have $\mu(\Gamma),\Delta \vdash t_i : \mu(\tau\prime_i)$. But we know that $\mu(\tau\prime_i) = \mu(\tau_i[\vec{\alpha} \mapsto \vec{\beta}])$, since $\mu \models \tau\prime_i \doteq \tau_i[\vec{\alpha} \mapsto \vec{\beta}]$, for all $i = 1,\ldots,n$. So we also have $\mu(\Gamma),\Delta \Vdash t_i : \mu(\tau_i[\vec{\alpha} \mapsto \vec{\beta}])$. Therefore, by a single application of the CPL rule, we get $\mu(\Gamma),\Delta \Vdash f(t_1,\ldots,t_n) : \mu(\tau[\vec{\alpha} \mapsto \vec{\beta}])$.

- If t is an equality $t_1 = t_2$, then we have $\Gamma,\Delta \vdash t_1 = t_2 : bool \mid \{t_1 = t_2\} \mid T_1 \cup T_2 \cup \{\tau_1 \doteq \tau_2\}$, given $\Gamma,\Delta \vdash t_1 : \tau_1\prime \mid \emptyset \mid T_1$ and $\Gamma,\Delta \vdash t_2 : \tau_2\prime \mid \emptyset \mid T_2$. We also know that $\mu \models T_1 \cup T_2 \cup \{\tau_1 \doteq \tau_2\}$, and any such μ is such that $\mu \models T_1$, $\mu \models T_2$, and $\mu \models \tau_1 \doteq \tau_2$. By the induction hypothesis, we have $\mu(\Gamma),\Delta \vdash t_1 : \mu(\tau_1)$ and $\mu(\Gamma),\Delta \vdash t_2 : \mu(\tau_2)$, but since $\mu \models \tau_1 \doteq \tau_2$, we know that $\mu(\tau_1) \equiv \mu(\tau_2)$. So by a single application of rule EQU, we get $\mu(\Gamma),\Delta \vdash t_1 = t_2 : \mu(bool)$, and $\mu(bool) = bool$.

$\square$

7.2 Constraint Solving

In this section we present a procedure that generalizes Robinson unification [18] to account for type constraints and produces solutions, where possible. Since each rule simplifies the constraints, together they induce a straightforward decision procedure for type and term constraints.

Suppose we want to unify two terms t_1 and t_2. Let us have $\Gamma,\Delta \vdash t_1 = t_2 : bool \mid C \mid T$ derived in the constraint generation step. Then we apply the following rewriting rules to the tuple (C,T), until none applies. We apply the rules in order, meaning we only apply rule n if no rule i with $i < n$ applies.

1. $(C,\{f(\tau_1,\ldots,\tau_n) \doteq f(\tau\prime_1,\ldots,\tau\prime_n)\} \cup Rest) \rightarrow (C,\{\tau_1 \doteq \tau\prime_1,\ldots,\tau_n \doteq \tau\prime_n\} \cup Rest)$
2. $(C,\{\tau \doteq \tau\} \cup Rest) \rightarrow (C, Rest)$
3. $(C,\{f(\tau_1,\ldots,\tau_n) \doteq g(\tau\prime_1,\ldots,\tau\prime_m)\} \cup Rest) \rightarrow wrong$, if $f \neq g$ or $n \neq m$
4. $(C,\{\tau \doteq \alpha\} \cup Rest) \rightarrow (C,\{\alpha \doteq \tau\} \cup Rest)$, if τ is not a type variable
5. $(C,\{\alpha \doteq \tau\} \cup Rest) \rightarrow (C,\{\alpha \doteq \tau\} \cup Rest[\alpha \mapsto \tau])$, if α does not occur in τ
6. $(C,\{\alpha \doteq \tau\} \cup Rest) \rightarrow wrong$, if α occurs in τ
7. $(\{f(t_1,\ldots,t_n) = f(s_1,\ldots,s_n)\} \cup Rest, T) \rightarrow (\{t_1 = s_1,\ldots,t_n = s_n\} \cup Rest, T)$
8. $(\{t = t\} \cup Rest, T) \rightarrow (Rest, T)$
9. $(\{f(t_1,\ldots,t_n) = g(s_1,\ldots,s_m)\} \cup Rest, T) \rightarrow false$, if $f \neq g$ or $n \neq m$
10. $(\{t = X\} \cup Rest, T) \rightarrow (\{X = t\} \cup Rest, T)$, if t is not a variable
11. $(\{X = t\} \cup Rest, T) \rightarrow (\{X = t\} \cup Rest[X \mapsto t], T)$, if X does not occur in t
12. $(\{X = t\} \cup Rest, T) \rightarrow false$, if X occurs in t.

We will use the symbol $\rightarrow^*$ to denote the reflexive and transitive closure of $\rightarrow$.

Example 6: Let $C = \{cons(X,[\,]) = cons(1,Y)\}$ and $T = \{\alpha_X = \nu,\ list(\gamma) = list(\nu), int = \eta,\ \alpha_Y = list(\eta), list(\nu) = list(\eta)\}$. Step-by-step the algorithm rewrite the pair (C,T) as follows:
$(C,T) \rightarrow (C,\{\alpha_X = \nu,\ \gamma = \nu, int = \eta,\ \alpha_Y = list(\eta), list(\nu) = list(\eta)\}) \rightarrow$
$(C,\{\alpha_X = \nu,\ \gamma = \nu, int = \eta,\ \alpha_Y = list(\eta), \nu = \eta\}) \rightarrow$
$(C,\{\alpha_X = \nu,\ \gamma = \nu, \eta = int,\ \alpha_Y = list(\eta), \nu = \eta\}) \rightarrow$
$(C,\{\alpha_X = \nu,\ \gamma = \nu, \eta = int,\ \alpha_Y = list(int), \nu = int\}) \rightarrow$
$(C,\{\alpha_X = \nu,\ \gamma = int, \eta = int,\ \alpha_Y = list(int), \nu = int\}(= T\prime)) \rightarrow$
$(\{X = 1, [\,] = Y\}, T\prime) \rightarrow (\{X = 1, Y = [\,]\}, T\prime)$

Note that, in the final pair, no more rules apply and we can interpret this pair as a pair of substitutions for terms and for types, respectively.

7.3 Properties of the Regular Typed Unification Algorithm

In this section we show the main properties of regular typed unification. Firstly, it always terminates. Secondly, it is correct, meaning that the result is the same as we would have gotten in the equality theory defined for $=$ semantically. One big obstacle for this second property is that terms may not be ground when we want to unify them, and semantically we always need a state to evaluate variables. We will be conservative and assume that if there is a possible state for which the terms have values in the same semantic domain, then there is no type error (yet). Similarly, if there is a state for which the terms have the same semantic value, then the result is not *false* (yet).

Theorem 3 - Termination: Let (C,T) be the sets of constraints generated for terms t_1 and t_2. The algorithm always terminates, returning a pair of unifiers, false, or wrong.

Proof: We divide the algorithm in two parts. The first consists of the rules 1 to 6, and the second of the rules 7 to 12. Each of these parts are the Martelli-Montanari algorithm [15] for its corresponding kind of constraints, type equality and equality, respectively. Therefore they terminate.

For a formal proof for the termination of the Martelli-Montanari algorithm, we defer the reader to [15].

Moreover, if the Martelli-Montanari terminates, the output is either a most general unifier, or the algorithm fails. In the first part, failure is represented by *wrong*, and in the second part, it is represented by *false*. So our algorithm either terminates and outputs *wrong*, *false*, or both parts succeed and the algorithm outputs a pair of most general unifiers. □

We now know that the algorithm terminates, and what the outputs might be. We will additionally prove that the result is semantically valid. We start by proving a few auxiliary lemmas.

The following lemmas are used to prove soundness.

Lemma 1 - Rewriting Consistency: Let $(C,T) \rightarrow (C\prime,T\prime)$ be a step in the typed unification algorithm, such that the output is not $false$ nor *wrong*. Then, if for all equality constraints $(t_1 = t_2) \in C\prime$ the substitution θ is a unifier of t_1 and t_2, then θ is also a unifier of each equality constraint in C. Same applies to $T\prime$ and T, with a type substitution μ.

Proof: We will prove this by case analysis.

1. $C\prime$ and C are equal so, trivially, any substitution θ that unifies each equality constraint in C also unifies each equality constraint in $C\prime$. Now suppose that μ is a type substitution such that $\mu(\tau_i) = \mu(\tau_i\prime)$, for $i = 1,\ldots,n$, then, also $\mu(f(\tau_1,\ldots,\tau_n)) = \mu(f(\tau\prime_1,\ldots,\tau\prime_n))$. All other type equality constraints in T are also in $T\prime$, so any unifier of $T\prime$ is a unifier of T.

2. $C\prime$ and C are equal so, trivially, any substitution θ that unifies each equality constraint in C also unifies each equality constraint in $C\prime$. All type equality constraints in $T\prime$ are also in T, so all unifiers of $T\prime$ are unifiers of that subset of T. Moreover, T has one more type equality constraint $\tau \doteq \tau$, but any substitution, in particular any unifier of $T\prime$ is also a unifier of τ with itself.

3. This case does not apply, since the output is *wrong*.

4. $C\prime$ and C are equal so, trivially, any substitution θ that unifies each equality constraint in C also unifies each equality constraint in $C\prime$. Any unifier of $T\prime$ is also a unifier of T, since swapping the terms on a type equality constraints does not change the fact that a substitution is a unifier.
5. $C\prime$ and C are equal so, trivially, any substitution θ that unifies each equality constraint in C also unifies each equality constraint in $C\prime$. Suppose μ is a unifier of $T\prime$, then $\mu(\alpha) = \mu(\tau)$. Therefore, since $T\prime = T[\alpha \mapsto \tau]$ for all constraints except $\alpha \doteq \tau$, then $\mu(T\prime) = \mu(T[\alpha \mapsto \tau]) = (\mu \circ [\alpha \mapsto \tau])(T)$ but since $\mu(\alpha) = \mu(\tau)$, then $(\mu \circ [\alpha \mapsto \tau])(T) = \mu(T)$. So μ is also a unifier of T.
6. This case does not apply, since the output is *wrong*.

The proof for the rest of the cases is similar to the proof for the cases 1 to 6, except we replace type equality constraints with equality constraints, type substitution with substitution, and *wrong* with $false$. □

Lemma 2 - Self-satisfiability: Suppose C is a set of equality constraints in normal form. Then, C can be interpreted as a substitution θ, and θ is a unifier of all constraints in C. Same can be said for a set of type equality constraints in normal form T.

Proof: If C is in normal form, then $C = \{X_1 = t_1, \dots, X_n = t_n\}$, where X_i is a variable and none of X_i occurs in any t_i. So, when we interpret C as a substitution θ, we will have $\theta = [X_1 \mapsto t_1, \dots, X_n \mapsto t_n]$. When we apply θ to each constraint in C, we will get $\theta(C) = \{\theta(X_1) = \theta(t_1), \dots, \theta(X_n) = \theta(t_n)\}$, but since none of the variables X_i occur in any t_i, then $\theta(t_i) = t_i$. Moreover, $\theta(X_i) = t_i$. So we get $\theta(C) = \{t_1 = t_1, \dots t_n = t_n\}$. Therefore, θ is a unifier of all constraints in C. The proof for type equality constraints is similar to this one, replacing substitutions with type substitutions and terms with type terms. □

We are now ready to prove the following theorem that proves that the algorithm outputs a semantically correct value.

Theorem 4 - Soundness of the Typed Unification Algorithm: Let t_1 and t_2 be the input to the typed unification algorithm, and $\Gamma \vdash t_1 = t_2 \mid C \mid T$. Suppose $(C,T) \to^* R$.

1. If $R = (\theta, \mu)$, a pair of substitutions for terms and types respectively, then $\theta, \mu \models C, T$.
2. If $R = false$, then there is no substitution θ such that $\theta \models C$, but there is a type substitution μ such that $\mu \models T$.
3. If $R = wrong$, then there is no type substitution μ such that $\mu \models T$.

Proof: The proof for (1) follows from Lemmas 1 and 2. We get that θ and μ are unifiers of C and T, respectively.

The proof for (2) follows from the fact that the Martelli-Montanari algorithm is complete, i.e., if there was a unifier for the equality constraint set C, then it would have been obtained. Therefore there is no unifier for C, so there is no θ such that $\theta \models C$. However, since we got to the second part of the algorithm, then we were able to find a unifier for the type equality constraints. This means that there is at least one unifier for T, so there is a μ such that $\mu \models T$.

The proof for (3) follows form the fact that the Martelli-Montanari algorithm is complete, i.e., if there was a unifier for the type equality constraint set T, then it would have been obtained. Therefore there is no unifier for T, so there is no μ such that $\mu \models T$. $\square$

We also prove that the unification algorithm outputs principal typings when it succeeds.

Theorem 5 - Completeness of the Typed Unification Algorithm: Let t be term, or a unification of two terms, Γ be a generic context, and Δ be type assumptions for constants and function symbols. If $\Gamma, \Delta \vdash t : \tau \mid C \mid T$ and $(C,T) \rightarrow^* (\theta, \mu)$. Then $(\mu(\Gamma), \mu(\tau))$ is a principal typing of t.

Proof: We will prove by structural induction on t.

- If t is a variable X, then $(X : \alpha) \in \Gamma$. We know that $\Gamma, \Delta \vdash X : \alpha \mid \emptyset \mid \emptyset$ by a single application of GVAR. $(\emptyset, \emptyset) \rightarrow ([\,],[\,])$, where $[\,]$ are each the identity substitution for variables and type variables, respectively. Therefore $[\,](\alpha) = \alpha$, and any type τ derived in the type system such that $(X : \tau) \in \Gamma\prime$, then τ is an instance of α.
- If t is a constant k, then $(k : \forall \vec{\alpha}.\tau) \in \Delta$. We know that $\Gamma, \Delta \vdash k : \tau[\vec{\alpha} \mapsto \vec{\sigma}]$, for any $\vec{\sigma}$. We get by a single application of rule GCST that $\Gamma, \Delta \vdash k : \tau[\vec{\alpha} \mapsto \vec{\beta}] \mid \emptyset \mid \emptyset$. $(\emptyset, \emptyset) \rightarrow ([\,],[\,])$, where $[\,]$ are each the identity substitution for variables and type variables, respectively. Therefore $[\,](\tau[\vec{\alpha} \mapsto \vec{\beta}]) = \tau[\vec{\alpha} \mapsto \vec{\beta}]$, and we known that any $\vec{\sigma}$ is an instance of $\vec{\beta}$.
- If t is a complex term $f(t_1, \dots, t_n)$, then $(f : \forall \vec{\alpha}.\tau_1 \times \cdots \times \tau_n \rightarrow \tau\prime) \in \Delta$. By the induction hypothesis, we know that if $(\emptyset, T_i) \rightarrow^* ([\,], \mu_i)$, then $(\mu_i(\Gamma), \mu_i(\tau\prime_i))$ is a principal typing of t_i. Now suppose $(\emptyset, T_1 \cup \cdots \cup T_n \cup \{\tau\prime_1 \doteq \tau_1[\vec{\alpha} \mapsto \vec{\beta}], \dots, \tau\prime_n \doteq \tau_n[\vec{\alpha} \mapsto \vec{\beta}]\}) \rightarrow^* ([\,], \mu)$. We know that $\mu(\tau\prime_i) = \mu(\tau_i[\vec{\alpha} \mapsto \vec{\beta}])$ for all $i = 1, \dots, n$. So we can derive $\mu(\Gamma), \Delta \vdash t_i : \mu(\tau_i[\vec{\alpha} \mapsto \vec{\beta}])$, and by a single application of rule CPL, we get $\mu(\Gamma), \Delta \vdash f(t_1, \dots, t_n) : \mu(\tau[\vec{\alpha} \mapsto \vec{\beta}])$. Now we need to prove that this typing $(\mu(\Gamma), \mu(\tau[\vec{\alpha} \mapsto \vec{\beta}]))$ is the principal typing. Suppose we had another typing that was not an instance of this one $(\mu\prime(\Gamma), \mu\prime(\tau[\vec{\alpha} \mapsto \vec{\beta}]))$. Since μ is an MGU of $T_1 \cup \cdots \cup T_n \cup \{\tau\prime_1 \doteq \tau_1[\vec{\alpha} \mapsto \vec{\beta}], \dots, \tau\prime_n \doteq \tau_n[\vec{\alpha} \mapsto \vec{\beta}]\}$, then either for some i $\mu\prime \not\models T_i$, or for some i $\mu\prime \not\models \tau\prime_i \doteq \tau_i[\vec{\alpha} \mapsto \vec{\beta}]$. If the former is true, then $(\mu\prime(\Gamma), \mu(\prime(\tau\prime_i)))$ is not an instance of the principal typing for t_i and therefore cannot be derived in the type system. If the latter is true, then $\mu\prime(\tau_i[\vec{\alpha} \mapsto \vec{\beta}]) \neq \mu\prime(\tau\prime_i)$ and we cannot use the rule CPL in the type system. Therefore, $(\mu(\Gamma), \mu(\tau[\vec{\alpha} \mapsto \vec{\beta}]))$ is the principal typing for $f(t_1, \dots, t_n)$.
- Suppose t is an equality $t_1 = t_2$. By the induction hypothesis, we know that if $(\emptyset, T_i) \rightarrow^* ([\,], \mu_i)$, then $(\mu_i(\Gamma), \mu_i(\tau_i))$ is a principal typing of t_i. Now suppose $(\{t_1 = t_2\}, T_1 \cup T_2 \cup \{\tau_1 \doteq \tau_2) \rightarrow^* (\theta, \mu)$. We know that $\mu(\tau_i)$ is an instance of $\mu_i(\tau_i)$, so we can derive $\mu(\Gamma), \Delta \vdash t_i : \mu(\tau_i)$ in the type system. By a single application of rule EQU, we get $\mu(\Gamma), \Delta \vdash t_1 = t_2 : bool$. So $(\mu(\Gamma), \mu(bool))$ is a typing. Suppose it was not the principal typing. Suppose we had another typing that was not an instance of this one $(\mu\prime(\Gamma), \mu\prime(bool))$. For all μ, $\mu(bool) = bool$. Since μ is an MGU of $T_1 \cup T_2 \cup \{\tau_1 \doteq \tau_2\}$, so if $\mu\prime$ is not an instance, then either for some i $\mu\prime \not\models T_i$ or $\mu\prime \not\models \tau_1 \doteq \tau_2$. If the former is true, then $(\mu\prime(\Gamma), \mu\prime(\tau_1))$ is not an instance of its principal typing, so it cannot be derived in the type system. If the latter is true, the types for t_1 and t_2 are different and we cannot apply rule EQU, so we could not have this derivation in the type system. Therefore, $(\mu(\Gamma), \mu(bool))$ is the principal typing for $t_1 = t_2$.

$\square$

8 Final Remarks

Our regular typed unification algorithm provides some foundation for the use of regular types to dynamically catch erroneous Prolog behaviors. Indeed, one of the original motivations for this work was to understand how to extend the YAP Prolog system [5] with an effective dynamic typing. In [3] we proposed a typed SLD-resolution (TSLD) which used our previous notion of typed unification. Our goal now is to effectively extend SLD resolution with unification of terms typed by deterministic regular types. A TSLD-tree branch may result in *true*, *false*, or *wrong*, depending on the same results for the unifications in the branch. In [3], each TSLD-tree branch where a unification outputs *false*, needed to continue execution on the same branch in order to check if there was a type error in some other atom in the query. This leads to a drastic increase in the runtime of programs.

Example 7: Consider the following (unrealistic) but possible program:

```
p(0).
```

and query: `?- p(1),...,p(900),p(a)`. In Prolog SLD-resolution the query fails after one SLD-step. In the TSLD-resolution defined in [3], since the first 900 queries return $false$, one needs to reach step 901 in order to obtain the value *wrong*.

We argue that when adding regular typed unification to Prolog we must have a compromise between completeness and efficiency. If the evaluation of a query is $false$ we stop execution, and the same happens for *wrong*. However, if the result is $false$ and there are other atoms in the query, we cannot assure that the value for that branch is indeed $false$, only that it is not $true$. Thus, in our extension to Prolog we output **no**(?) in these cases. On the other hand, we output **no**(false) if there are no other atoms in the query, and **no**(wrong) if the branch ends on *wrong*. Note that in many programs, for some queries, we are *always* able to detect the type error.

Example 8: Consider the predicate that calculates the length of a list:

```
length([], 0).
length([_|T],N) :- length(T,N1), N is N + 1.
```

One typical bug is to swap the arguments of a predicate. Now note that, in this case, if we have the erroneous query `?- length(3,[a,b,c])`, both branches of the TSLD-tree output *wrong* since there is a type error in the first argument (and also in the second).

9 Related Work

This paper generalizes a typed unification algorithm previously defined by the authors in [3] that was used in the dynamic typing of logic programs. In [3], functions symbols f of arity n had co-domains which were always sets of terms of the form $f(t_1,\ldots,t_n)$, where the arguments t_i belong to the corresponding domain of f. Here we extend this notion enabling the use of semantic domains and co-domains described by deterministic regular types allowing a non-Herbrand interpretation of function symbols.

The most obvious related work is many-sorted unification [23], though many-sorted unification assumes an infinite hierarchy of sorts and we do not assume a hierarchy of types. In particular there is a relation with many-sorted unification with a forest-structured sort hierarchy [23], but even compared with this strong restricted unification problem, our work gives easier and nicer results, mostly due to the use of an expressive universe partition based on regular types but with no underlying hierarchy on the domains.

Here we study unification of terms interpreted in domains described by regular types, and we allow a form of parametric polymorphism in the description of term variables. Parametric polymorphic descriptions of sorted domains goes back to Smolka generalized order-sorted logic [22]. In his system, subsort declarations are propagated to complex type expressions, thus the main focus is on subtyping which is not the scope of our work.

Dart and Zobel [7] provided an algorithm for regular type unification, generating a type unifier. Due to problems related to tuple distributivity, not all types had a most general type unifier. In consequence, unification returned a weak type unifier. However, the question whether unification returned a minimal weak type unifier was unknown and left as an open question.

In [11], there is a typed unification algorithm used in a typed operational semantics for logic programming. The main difference to our work is that in [11] failing unification due to ill-typedness is not detected with a different value and it is not different from a well-typed failed unification.

A data type reconstruction algorithm was previously defined in [19] based on equations and inequations constraints. This was also applied to logic programs (terms and predicates). Here we focus on term unification, thus equality is the only predicate, and this rather simplifies our type system.

Acknowledgements This work was partially financially supported by UIDB/00027/2020 of the Artificial Intelligence and Computer Science Laboratory, LIACC, funded by national funds through the FCT/MCTES (PIDDAC).

References

[1] Krzysztof R. Apt (1996): *From Logic Programming to Prolog*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[2] João Barbosa, Mário Florido & Vítor Santos Costa (2021): *Data Type Inference for Logic Programming*. In: *Logic-Based Program Synthesis and Transformation*, LNCS, Springer, pp. 16–37, doi:10.1007/978-3-030-98869-2_2.

[3] João Barbosa, Mário Florido & Vítor Santos Costa (2022): *Typed SLD-Resolution: Dynamic Typing for Logic Programming*. In: *Logic-Based Program Synthesis and Transformation*, LNCS, Springer, doi:10.1007/978-3-031-16767-6_7.

[4] Michael Codish & Vitaly Lagoon (2000): *Type dependencies for logic programs using ACI-unification*. *Theor. Comput. Sci.* 238(1-2), pp. 131–159, doi:10.1016/S0304-3975(98)00122-4.

[5] Vítor Santos Costa, Ricardo Rocha & Luís Damas (2012): *The YAP Prolog System*. *Theory and Practice of Logic Programming* 12(1-2), pp. 5–34, doi:10.1017/S1471068411000512.

[6] Luís Damas & Robin Milner (1982): *Principal Type-Schemes for Functional Programs*. In: *Conference Record of the Ninth Annual ACM Symposium on Principles of Programming Languages*, pp. 207–212, doi:10.1145/582153.582176.

[7] Philip W. Dart & Justin Zobel (1992): *A Regular Type Language for Logic Programs*. In Frank Pfenning, editor: *Types in Logic Programming*, The MIT Press, pp. 157–187.

[8] Pierre Deransart, AbdelAli Ed-Dbali & Laurent Cervoni (1996): *Prolog - the standard: reference manual*. Springer, doi:10.1007/978-3-642-61411-8.

[9] Thom W. Frühwirth, Ehud Y. Shapiro, Moshe Y. Vardi & Eyal Yardeni (1991): *Logic Programs as Types for Logic Programs*. In: Proc. of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Netherlands,1991, pp. 300–309, doi:10.1109/LICS.1991.151654.

[10] John P Gallagher & Kim S Henriksen (2004): *Abstract Domains Based on Regular Types*. In: International Conference on Logic Programming, Springer, pp. 27–42, doi:10.1007/978-3-540-27775-0_3.

[11] Michael Hanus (1991): *Horn clause programs with polymorphic types: semantics and resolution*. Theoretical Computer Science 89(1), pp. 63–106, doi:10.1016/0304-3975(90)90107-S. Available at `https://www.sciencedirect.com/science/article/pii/030439759090107S`.

[12] Manuel V. Hermenegildo, José F. Morales, Pedro López-García & Manuel Carro (2023): *Types, Modes and so Much More - The Prolog Way*. In: Prolog: The Next 50 Years, LNCS 13900, Springer, pp. 23–37, doi:10.1007/978-3-031-35254-6_2.

[13] Trevor Jim (1996): *What Are Principal Typings and What Are They Good For?* In: POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, pp. 42–53, doi:10.1145/237721.237728.

[14] J. W. Lloyd (1984): *Foundations of Logic Programming*. Springer-Verlag, Berlin, Heidelberg, doi:10.1007/978-3-642-96826-6.

[15] Alberto Martelli & Ugo Montanari (1982): *An Efficient Unification Algorithm*. ACM Trans. Program. Lang. Syst. 4(2), p. 258–282, doi:10.1145/357162.357169.

[16] Robin Milner (1978): *A Theory of Type Polymorphism in Programming*. J. Comput. Syst. Sci. 17(3), pp. 348–375, doi:10.1016/0022-0000(78)90014-4.

[17] Prateek Mishra (1984): *Towards a Theory of Types in Prolog*. In: Proceedings of the 1984 International Symposium on Logic Programming, IEEE-CS, pp. 289–298.

[18] J. A. Robinson (1965): *A Machine-Oriented Logic Based on the Resolution Principle*. Journal of the ACM 12(1), pp. 23–41, doi:10.1145/321250.321253.

[19] Tom Schrijvers & Maurice Bruynooghe (2006): *Polymorphic algebraic data type reconstruction*. In: Proceedings of the 8th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP, doi:10.1145/1140335.1140347.

[20] Tom Schrijvers, Maurice Bruynooghe & John P. Gallagher (2008): *From Monomorphic to Polymorphic Well-Typings and Beyond*. In: Logic-Based Program Synthesis and Transformation, 18th International Symposium, LOPSTR, LNCS 5438, Springer, doi:10.1007/978-3-642-00515-2_11.

[21] Tom Schrijvers, Vítor Santos Costa, Jan Wielemaker & Bart Demoen (2008): *Towards Typed Prolog*. In: Logic Programming, 24th International Conference, ICLP, LNCS 5366, Springer, pp. 693–697, doi:10.1007/978-3-540-89982-2_59.

[22] Gert Smolka (1988): *Logic Programming with Polymorphically Order-Sorted Types*. In: Algebraic and Logic Programming, International Workshop, LNCS 343, Springer, pp. 53–70, doi:10.1515/9783112620267-007.

[23] Christoph Walther (1988): *Many-Sorted Unification*. J. ACM 35(1), p. 1–17, doi:10.1145/42267.45071.

[24] M. Wand (1987): *A simple algorithm and proof for type inference*. Fundamenta Infomaticae 10, pp. 115–122, doi:10.3233/FI-1987-10202.

[25] Eyal Yardeni & Ehud Shapiro (1991): *A type system for logic programs*. The Journal of Logic Programming 10(2), pp. 125–153, doi:10.1016/0743-1066(91)80002-U. Available at `https://www.sciencedirect.com/science/article/pii/074310669180002U`.

[26] Justin Zobel (1987): *Derivation of Polymorphic Types for Prolog Programs*. In: Logic Programming, Proceedings of the Fourth International Conference, ICLP.

Order-Sorted Intensional Logic: Expressing Subtyping Polymorphism with Typing Assertions and Quantification over Concepts*

Đorđe Marković   Marc Denecker

KU Leuven, Department of computer science, Leuven, Belgium

dorde.markovic@kuleuven.be   marc.denecker@kuleuven.be

Subtyping, also known as subtype polymorphism, is a concept extensively studied in programming language theory, delineating the substitutability relation among datatypes. This property ensures that programs designed for supertype objects remain compatible with their subtypes.

In this paper, we explore the capability of order-sorted logic for utilizing these ideas in the context of Knowledge Representation. We recognize two fundamental limitations: First, the inability of this logic to address the concept rather than the value of non-logical symbols, and second, the lack of language constructs for constraining the type of terms. Consequently, we propose guarded order-sorted intensional logic, where guards are language constructs for annotating typing information and intensional logic provides support for quantification over concepts.

1 Introduction

The logic-based approach to knowledge representation (KR) dates back to the early ages of artificial intelligence. From the inception of this approach, limitations of untyped logic were identified. These issues led to the use of *many-sorted logic* [10], and *order-sorted logic* (*OSL*) [1]. In *many-sorted logic*, the domain of discourse (or universe) is partitioned into different sorts/types, all disjoint. The latter assumption is lifted in *order-sorted logic*, and sorts/types can be organized in a hierarchy by inclusion.

When extending first-order logic with ordered sorts, the concept of *subtyping polymorphism* emerges [8, Chapter 15]. A prime example of this concept is the modeling of characteristic behaviors among different animals. Dogs bark, cats meow, etc., while nearly all animals produce species-specific sounds. In this scenario, *animal* serves as the overarching type, with specific animal types acting as subtypes. In many programming languages, one can invoke a method such as *produce sound* for an animal, which *dynamically dispatches* behavior based on the specific species of the animal. Logic is characterized by model semantics, and hence, it lacks the notion of method invocation found in programming languages. Nonetheless, logical statements can draw inspiration from this concept. For instance, consider the statement: *"There is an animal in my yard that is either barking or meowing"*. Considering that *barking* and *meowing* are predicates defined respectively on types *dog* and *cat* which are subtypes of the type *animal*, expressing such statements in *OSL* may easily lead to untyped formulae, as we shall see later.

In this paper, we explore the principles underlying subtyping polymorphism and highlight challenges in its representation within order-sorted first-order logic. Additionally, we identify the two key principles essential for naturally expressing such concepts in any logic employing order-sorts. The first principle relates to the inherent incapacity of standard *OSL* to condition the subtyping relation of a term. For

*This work was partially supported by the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen".

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 253–266, doi:10.4204/EPTCS.416.22

instance, given a variable x ranging over type *animal*, it is impossible[1] to express *"if x is of type dog then x is barking"*. Furthermore, we show that making such typing assertions implicit (annotated) is essential for subtyping. Whereby, annotating aims to implicitly constrain the type of a variable to the type of the argument it occurs at, given that the type of the variable is a supertype of the argument. For example, a language can be extended such that the statement *"$\langle\langle$x is barking$\rangle\rangle$"* stands for *"if x is of type dog then x is barking"*. This is possible because predicate *barking* caries the typing information that an argument of type *dog* is expected. The second principle tackles the constraint of first-order logic to solely address the values (extensions) of non-logical symbols rather than the concepts (intensions) interpreting them and the constraint to quantify over these concepts. Principles of intensional logic ([4]) become crucial in overcoming these limitations. We demonstrate that by combining order-sorted logic with principles of intensional logic and introducing the innovative principle of implicit type conditions, we establish a novel language suitable for expressing subtyping polymorphism.

The remainder of the paper is structured as follows: (2) Order-sorted logic preliminaries; (3) Analysis of subtyping polymorphism from the logic perspective; (4) Introduction of guarded and (5) intensional *OSL*; (6) Presentation of results: Guarded order-sorted intensional logic; (7) Discussion on well-typedness in order-sorted intensional logic; (8) Semantics of the language; (9) Discussion of related work; (10) Conclusion.

2 Preliminaries – Order-sorted Logic

This section formally defines *order-sorted logic*. We start with the notion of a vocabulary.

Definition 1. *An **OSL vocabulary** Σ of non-logical symbols is a quadruple (T_Σ, S_Σ, $<:_\Sigma$, ts_Σ) where:*

- *T_Σ is a set of **type symbols** $\mathbb{T}$. Type symbols $\mathbb{U}$ (universe), $\mathbb{B}$ (boolean), and $\mathbb{N}$ (natural numbers) are always member of T_Σ.*
- *S_Σ is a set of **function and predicate** symbols.*
- *$<:_\Sigma$ is a **subtyping** relation on T_Σ. Type $\mathbb{S}$ is a **direct subtype** of $\mathbb{T}$ if $\mathbb{S} <:_\Sigma \mathbb{T}$. For each type $\mathbb{T}$ (except $\mathbb{U}$) without direct supertype declaration $\mathbb{T} <:_\Sigma \mathbb{S}$, we implicitly assume $\mathbb{T} <:_\Sigma \mathbb{U}$. Accordingly, $\mathbb{B} <:_\Sigma \mathbb{U}$ and $\mathbb{N} <:_\Sigma \mathbb{U}$.*
- *ts_Σ is a **type signature** associating to every symbol in S_Σ a word of the following format $(\mathbb{T}_1 \times \cdots \times \mathbb{T}_n) \rightarrow \mathbb{T}$ (i.e., type term). If $\mathbb{T} = \mathbb{B}$, the symbol is a predicate symbol, otherwise it is a function symbol. The sets of predicate and function symbols are denoted with S^p_Σ, respectively S^f_Σ.*

*Type $\mathbb{T}_1$ is called a **subtype** of $\mathbb{T}_2$ if there is a path from $\mathbb{T}_1$ to $\mathbb{T}_2$ in the relation $<:_\Sigma$.*

Proposition 1. *Given vocabulary Σ, every type $\mathbb{T}$ in T_Σ (except $\mathbb{U}$) is subtype of $\mathbb{U}$.*

Proof. Follows directly from Definition 1. □

A symbol with type term $() \rightarrow \mathbb{T}$ is an object (or a constant function) symbol. A symbol with type term $() \rightarrow \mathbb{B}$ is a propositional symbol.

[1] Here we strictly talk about the incapability to constrain the subtype of a variable, and not about possible alternative modelings that would circumvent this issue by changing the ontology.

Example 1. *The following vocabulary declares: types $\mathbb{A}nimal$ (Animal), $\mathbb{C}at$ (Cat), and $\mathbb{D}og$ (Dog), where cat and dog are subtypes of animal; function age mapping animals to natural numbers; constant tom of type cat; and two predicates bark and meow denoting sets of dogs and cats respectively.*

$$\text{type } \mathbb{A}nimal \qquad \text{type } \mathbb{C}at <: \mathbb{A}nimal \qquad \text{type } \mathbb{D}og <: \mathbb{A}nimal$$
$$age : \mathbb{A}nimal \to \mathbb{N} \qquad tom : () \to \mathbb{C}at \qquad bark : \mathbb{D}og \to \mathbb{B} \qquad meow : \mathbb{C}at \to \mathbb{B}$$

The following defines *OSL* terms, formulae, expressions, and sentences.

Definition 2. *Given an infinite set X of variable symbols and OSL vocabulary Σ, an OSL **term** (τ) and **formula** (ϕ) over X and Σ are defined inductively (using BNF):*

- *A **term** is a variable or a function term:*
  $\langle\tau\rangle ::= \langle x\rangle \mid \langle f\rangle(\langle\tau\rangle,..., \langle\tau\rangle)$ *where* $x \in X, f \in S_\Sigma^f$
- *A **formula** is true or false, an atomic formula, a negation, a disjunction, or an existential quantification:*
  $\langle\phi\rangle ::= \textbf{\textit{true}} \mid \textbf{\textit{false}} \mid \langle p\rangle(\langle\tau\rangle,...,\langle\tau\rangle) \mid \neg\langle\phi\rangle \mid \langle\phi\rangle \vee \langle\phi\rangle \mid \exists\langle x\rangle\ [\langle\mathbb{T}\rangle] : \langle\phi\rangle$
  where $x \in X, p \in S_\Sigma^p, \mathbb{T} \in T_\Sigma$
- *An OSL **expression** (α) is either an OSL* term *or an OSL* formula*. A formula with no free variables (all variables in the formula are quantified) is a **sentence**.*

Other familiar connectives, $\wedge$, $\Rightarrow$, $\forall$, and $\Leftrightarrow$ can be defined in the standard way as shortcuts in terms of the basic ones. Furthermore, we assume the language is equipped with the standard set of predicates and functions, i.e., $=$ for each type and standard arithmetic operations ($+, -, \times, \dots$) on natural numbers.

Example 2. *An example of a term: $age(tom)$; formula: $age(x) = 15$; and sentence: $\exists a[\mathbb{A}nimal] : age(a) = 15$.*

Sentence $bark(tom)$ is well-formed according to Definition 2. However, the typing information shows that it is senseless as cats cannot bark. For that reason, it is customary to define a syntactic subclass of Definition 2 that avoids such category clashes. These are the well-typed formulae.

Definition 3. *Given an infinite set X of variable symbols and an OSL vocabulary Σ, a **typing context** ω is a set of typing annotations of the format $s : t$ where s is a symbol from $X \cup S_\Sigma$ and t is a type term over T_Σ. A **typing relation** $\omega \vdash \alpha : \mathbb{T}$, meaning that expression α is of type $\mathbb{T}$ in the context ω, is defined by the following inductive definition:*

$$\frac{}{\omega \vdash \textbf{\textit{true}} : \mathbb{B}}\ \textit{(T-tr)} \qquad \frac{}{\omega \vdash \textbf{\textit{false}} : \mathbb{B}}\ \textit{(T-fa)} \qquad \frac{\omega \vdash \phi : \mathbb{B} \qquad \omega \vdash \varphi : \mathbb{B}}{\omega \vdash (\phi \vee \varphi) : \mathbb{B}}\ \textit{(T-or)}$$

$$\frac{\omega \vdash \phi : \mathbb{B}}{\omega \vdash \neg\phi : \mathbb{B}}\ \textit{(T-neg)} \qquad \frac{\omega \cup \{x : \mathbb{T}\} \vdash \phi : \mathbb{B}}{\omega \vdash (\exists x[\mathbb{T}] : \phi) : \mathbb{B}}\ \textit{(T-ex)} \qquad \frac{\omega \vdash t : \mathbb{S} \qquad \mathbb{S} <:_\Sigma \mathbb{T}}{\omega \vdash t : \mathbb{T}}\ \textit{(T-sub)}$$

$$\frac{x : \mathbb{T} \in \omega}{\omega \vdash x : \mathbb{T}}\ \textit{(T-var)} \qquad \frac{s : \mathbb{T}_1 \times \cdots \times \mathbb{T}_n \to \mathbb{T} \in \omega \qquad \omega \vdash t_1 : \mathbb{T}_1 \ \dots\ \omega \vdash t_n : \mathbb{T}_n}{\omega \vdash s(t_1, \dots, t_n) : \mathbb{T}}\ \textit{(T-app)}$$

*An OSL expression α over OSL vocabulary Σ with free variables $x_1, \dots, x_n$ is **well-typed** iff there are types $\mathbb{T}_1, \dots, \mathbb{T}_n, \mathbb{T} \in T_\Sigma$ such that $\omega = \{s : t \mid (s \in S_\Sigma \text{ and } ts_\Sigma(s) = t) \text{ or } (s \text{ is } x_i \text{ and } t \text{ is } \mathbb{T}_i \text{ for } i \in 1 \dots n)\}$ and $\omega \vdash \alpha : \mathbb{T}$. An OSL sentence ψ is well-typed iff $\omega \vdash \psi : \mathbb{B}$ where $\omega = \{s : t \mid s \in S_\Sigma \text{ and } ts_\Sigma(s) = t\}$ (as ψ has no free variables).*

The rules in this definition (a.k.a. typing judgments) define the type of an expression in a context (below the line) given that certain conditions (above the line) are satisfied. The specific rules are existential quantification (T-EX) which introduces new typing annotation to the context[2], and subtyping rule (T-SUB) which expresses that the term of type $\mathbb{S}$ can be seen as of type $\mathbb{T}$ if it holds that $\mathbb{T}$ is a supertype of $\mathbb{S}$. For an *OSL* sentence to be well-typed, the context initially has to correspond to the type signature of the function and predicate symbols from the vocabulary (ts_Σ). Given the vocabulary from Example 1, the formula $bark(tom)$ is ill-typed (i.e., not well-typed) because predicate *bark* expects argument of type $\mathbb{D}og$ and *tom* is of type $\mathbb{C}at$. In general and informally, a formula is well-typed if the type of each expression occurring as an argument to a function/predicate symbol is a subtype or of the same type as the type of that argument.

3 Analysis of subtyping polymorphism

As previously noted, the statement $bark(tom)$ is considered unacceptable (ill-typed) due to the category clash it contains. Specifically, barking does not apply to cats. One might argue that such statement could be accepted if always interpreted as false, thereby justifying its meaning as "Tom is a dog and $bark(tom)$". Since Tom is not a dog, the statement is false. But what then is the meaning of $\neg bark(tom)$? If it is interpreted as "Tom is a dog and $\neg bark(tom)$", then this formula is false, violating the law of excluded middle. An alternative interpretation is "$\neg$(Tom is a dog and $bark(tom)$)", in which case the formula is true, which seems to be a more reasonable choice in this case. However, notice that this statement is equivalent to "If Tom is a dog then $\neg bark(tom)$".

This brings us to an alternative interpretation of ill-typed formulae. One could argue that the initial formula $bark(tom)$ should be interpreted as "If Tom is a dog then $bark(tom)$". Consequently, it is justified to assert that statement $bark(tom)$ carries ambiguity, and hence can be considered as potentially dangerous, and therefore should be rejected (corresponding to a well-typed criterion). However, we argue that extending *OSL* language to support explicitly disambiguated forms of these ill-typed formulae is beneficial. We demonstrate this in the remainder of the section.

Consider the definition of the predicate $makingSound : \mathbb{A}nimal \to \mathbb{B}$ representing the set of all animals producing their specific sound. In the running example cats and dogs. This can be formalized in *OSL* as:

$$\forall a[\mathbb{A}nimal] : makingSound(a) \Leftrightarrow \begin{pmatrix} (\exists c[\mathbb{C}at] : a \overset{\mathbb{A}nimal-\mathbb{C}at}{=\!=} c \wedge meow(c)) \vee \\ (\exists d[\mathbb{D}og] : a \overset{\mathbb{A}nimal-\mathbb{D}og}{=\!=} d \wedge bark(d)) \end{pmatrix} \tag{1}$$

Note that equalities $\overset{\mathbb{A}nimal-\mathbb{C}at}{=\!=}$ and $\overset{\mathbb{A}nimal-\mathbb{D}og}{=\!=}$ are necessary since they operate on different types. Returning to the main point, in this example, it would be beneficial to constrain the type of variable a which ranges over type $\mathbb{A}nimal$ in the following way.

$$\forall a[\mathbb{A}nimal] : makingSound(a) \Leftrightarrow \big((\mathbb{C}at(a) \wedge meow(a)) \vee (\mathbb{D}og(a) \wedge bark(a))\big) \tag{2}$$

Similarly, the statement "all animals produce their specific sound" could be expressed as:

$$\forall a[\mathbb{A}nimal] : \big((\mathbb{C}at(a) \Rightarrow meow(a)) \wedge (\mathbb{D}og(a) \Rightarrow bark(a))\big) \tag{3}$$

However, these do not constitute *OSL* formulae as types are used as predicates and variable a of type $\mathbb{A}nimal$ remains an argument of predicates *meow* and *bark*. Notice that there is room for improvement

[2]Alternatively, one could say that this rule projects away typing information, depending on whether the rule is interpreted top-down or bottom-up.

in the statements above. Specifically, capability to talk about "sounds specific" for an animal would enhance the expressivity of the language.

Accordingly, the first goal of this paper is to extend order-sorted logic by introducing new language constructs (*guards*) as motivated in this section. The next step is to make these guards implicit, so it is possible to express $\mathbb{C}at(a) \Rightarrow meow(a)$ as $\langle\langle meow(a)\rangle\rangle$. Finally, to be able to talk about "sounds specific" for an animal the language needs to be extended with the intensional logic. These are presented in Section 4 and 5.

4 Guarded order-sorted logic

The extension of *OSL* with the concept of guarding terms by typing assertions is characterized in the following definition.

Definition 4. *Definition 1 of an OSL vocabulary Σ is extended with the following rule: if $\mathbb{T}$ is a type symbols in T_Σ, then $\mathbb{T} \in S_\Sigma$ and $ts_\Sigma(\mathbb{T}) = \mathbb{U} \to \mathbb{B}$.*

Definition 3 of an OSL typing relation, is extended with the two new rules, namely conjunction guarding (G-c) and implication guarding (G-i):

$$\frac{\omega \overset{n}{\underset{i=1}{\vdash}} t_i : \mathbb{U} \quad \omega \overset{n}{\underset{i=1}{\cup}} \{t_i : \mathbb{T}_i\} \vdash \phi : \mathbb{B}}{\omega \vdash (\mathbb{T}_1(t_1) \wedge \cdots \wedge \mathbb{T}_n(t_n) \wedge \phi) : \mathbb{B}}\ \textit{(G-c)} \qquad \frac{\omega \overset{n}{\underset{i=1}{\vdash}} t_i : \mathbb{U} \quad \omega \overset{n}{\underset{i=1}{\cup}} \{t_i : \mathbb{T}_i\} \vdash \phi : \mathbb{B}}{\omega \vdash (\mathbb{T}_1(t_1) \wedge \cdots \wedge \mathbb{T}_n(t_n) \Rightarrow \phi) : \mathbb{B}}\ \textit{(G-i)}$$

Example 3. *In the guarded OSL, statement "There is an animal (that is a cat) meowing!" can be expressed as: $\exists a[\mathbb{A}nimal] : \mathbb{C}at(a) \wedge meow(a)$. Towards making the judgment that this formula is well-typed (i.e., of type $\mathbb{B}$), let the context ω correspond to the typing signature of vocabulary from Example 1:*

$$\omega = \{\mathbb{C}at : \mathbb{U} \to \mathbb{B};\ \mathbb{D}og : \mathbb{U} \to \mathbb{B};\ age : \mathbb{U} \to \mathbb{N}; \ldots\ meow : \mathbb{C}at \to \mathbb{B}\}$$

For compact representation of derivation we use the following abbreviations:

$$\omega' = \omega \cup \{a : \mathbb{A}nimal\} \qquad\qquad \omega'' = \omega' \cup \{a : \mathbb{C}at\}$$

The following derivation provides the judgment that this formula is well-typed:

$$\dfrac{\dfrac{\dfrac{\checkmark_1}{\omega' \vdash a : \mathbb{A}nimal} \quad \dfrac{\checkmark_2}{\mathbb{A}nimal <:_\Sigma \mathbb{U}}}{\omega' \vdash a : \mathbb{U}}\ \textit{T-sub} \qquad \dfrac{\dfrac{\checkmark_3}{meow : \mathbb{C}at \to \mathbb{B} \in \omega''} \quad \dfrac{\checkmark_4}{a : \mathbb{C}at \in \omega''}}{\omega'' \vdash meow(a) : \mathbb{B}}\ \textit{T-app}}{\dfrac{\omega' \vdash \mathbb{C}at(a) \wedge meow(a) : \mathbb{B}}{\omega \vdash \exists a[\mathbb{A}nimal] : \mathbb{C}at(a) \wedge meow(a) : \mathbb{B}}\ \textit{T-ex}}\ \textit{G-c}$$

The justification for each of the final premises ($\checkmark$) is:

$\checkmark_1$ $a : \mathbb{A}nimal \in \omega'$ *since* $\omega' = \omega \cup \{a : \mathbb{A}nimal\}$.

$\checkmark_2$ *Since* $\mathbb{A}nimal$ *has no supertype in* Σ, *it follows that* $\mathbb{A}nimal <:_\Sigma \mathbb{U}$ *(Definition 1).*

$\checkmark_3$ $meow : \mathbb{C}at \to \mathbb{B} \in \omega''$ *since it is in* ω *and* $\omega'' = \omega \cup \{a : \mathbb{A}nimal\} \cup \{a : \mathbb{C}at\}$.

$\checkmark_4$ $a : \mathbb{C}at \in \omega''$ *since* $\omega'' = \omega' \cup \{a : \mathbb{C}at\}$ *(due to Definition 4, rule* (G-C)*).*

Further, it is possible to make these typing annotations implicit by introducing new language constructs.

Definition 5. *Let ψ be an OSL formula, ω a typing context, and $\{t_1,\dots,t_n\}$ terms in ψ (over OSL vocabulary Σ) such that: (1) $\omega \vdash t_i : \mathbb{T}_i$; (2) t_i occurs in ψ as an argument of predicate/function that expects argument of type $\mathbb{S}_i$; (3) $\mathbb{S}_i <:_\Sigma \mathbb{T}_i$; then:*

$$\begin{array}{lll} [[\psi]] & \textit{stands for} & \mathbb{S}_1(t_1) \wedge \cdots \wedge \mathbb{S}_n(t_n) \wedge \psi \\ \langle\langle\psi\rangle\rangle & \textit{stands for} & \mathbb{S}_1(t_1) \wedge \cdots \wedge \mathbb{S}_n(t_n) \Rightarrow \psi \end{array}$$

Example 4. *Employing implicit guarding, Example 3 becomes:* $\exists a[\mathbb{A}nimal] : [[meow(a)]]$.

5 Order-sorted intensional logic

The main concern of intensional logic is the difference between a concept (or intension) and, its value (or extension) in a state of affairs. A prototypical example is the "morning star" and "evening star", which represent distinct concepts (respectively, the star in the east before sunrise, and the star in the west after sunset), while denoting the same object in the actual state of affairs (the planet Venus). In the computational intensional logic of [2], intensions of predicates are first class objects that can be quantified over and stored in other predicates. For example, given a predicate *humanDisease* containing a set of intensions of unary predicates over humans (e.g., *flu*, *measels*, …) and type $\mathbb{C}$ representing all concepts, one can define *healtyHuman* as:

$$\forall h[\mathbb{H}uman] : (healthyHuman(h) \Leftrightarrow \neg\exists c[\mathbb{C}] : humanDisease(c) \wedge \$(c)(h))$$

Here $\$(c)$ is the value of the intensional object c. A similar approach can be applied to improve the formula (2) from Section 3; here *sound* is a unary predicate over animal sound intensions (in the running example *meow* and *bark*):

$$\forall a[\mathbb{A}nimal] : makingSound(a) \Leftrightarrow \exists c[\mathbb{C}] : sound(c) \wedge \$(c)(a) \tag{4}$$

However, this formula has a typing issue since sound concepts (variable c) can not be applied to an arbitrary animal (variable a), which is done by $\$(c)(a)$. This issue will be addressed after we formally introduce ordered-sorted intensional logic. First, a new built-in type $\mathbb{C}$ representing the set of concepts of the vocabulary is added to the *OSL* vocabulary. This type represents the collection of all symbols (types, functions, and predicates) within the vocabulary. The concept associated with a symbol s is denoted by $\widetilde{s}$ and can be accessed with the reference operator $`(s)$. The dual dereference operator $\$(\widetilde{s})$ is a unary higher-order function that, given a concept $\widetilde{s}$, returns the function or predicate associated with the symbol s. Therefore, $\$(\widetilde{s})$ is always followed by another bracket containing a tuple of terms that are applied to the resulting function or predicate. Accordingly, these terms should match the type and arity of the symbol. Formally:

Definition 6. *The order-sorted intensional logic is defined by the following extensions:*

1. *An OSL vocabulary Σ contains the build-in type $\mathbb{C}$ (concepts).*
2. *Type $\mathbb{C}$ denotes the set of all concepts in the vocabulary $\{\widetilde{s} \mid s \in S_\Sigma \cup T_\Sigma\}$.*
3. *Given an OSL vocabulary Σ, for $s \in S_\Sigma \cup T_\Sigma$, $`(s)$ is a term of type $\mathbb{C}$.*
4. *If term c is of type $\mathbb{C}$ then $\$(c)(\bar{t})$ is an OSL expression, where $\bar{t}$ is a tuple of terms.*

Example 5. *In the running example, type $\mathbb{C}$ denotes the set $\{\widetilde{\mathbb{B}}, \widetilde{\mathbb{N}}, \widetilde{\mathbb{A}nimal}, \widetilde{\mathbb{C}at}, \widetilde{\mathbb{D}og}, \widetilde{age}, \widetilde{tom}, \widetilde{bark}, \widetilde{meow}\}$. Type $\mathbb{S}ound$ (sounds) of animals can be declared as:*

$$\text{type } \mathbb{S}ound <: \mathbb{C} := \{`(meow), `(bark)\}$$

Notation $\mathbb{T} := \{\dots\}$ declares extension of type $\mathbb{T}$. Term $`(meow)$ denotes the concept $\widetilde{meow}$. An example of a formula is: $meow(\$(`tom)())$, which is the same as: $meow(tom)$.

Consider the statement $bark(\$(`tom)())$. It is a well-formed formula according to the Definition 6. It expresses that the extension of the intension of *tom* is barking, which is a complex way to say that *tom* is barking, i.e., $bark(tom)$. The utility of this sort of expression will become apparent only in a few paragraphs. However, here is important to notice that this statement is not well-typed, as *tom* is of type $\mathbb{C}at$ and *bark* is a predicate expecting an argument of type $\mathbb{D}og$. Furthermore, the Definition 3 (well-typed formulae), does not account for intensional logic. The criteria for well-formedness and well-typedness of a formula becomes challenging in intensional logic. This is because these properties become dependent on the extensions of types and other symbols (for more details see Section 7). For this paper, it suffices to reinstate these criteria by verifying the *grounded* version of a formula. Grounding a variable in a formula involves substituting it with individuals from the domain of its type. Additionally, intensional terms of the form $`(s)$ are grounded to $\widetilde{s}$ and intensional application $\$(\widetilde{s})(\bar{t})$ to $s(\bar{t})$ (here s is a symbol form a vocabulary). We demonstrate this idea on the following example.

Example 6. *Consider the following formalization (using the type $\mathbb{S}ound$) of the statement from formula (4): "An animal is making sound iff there is a sound it is producing".*

$$\forall a[\mathbb{A}nimal] : makingSound(a) \Leftrightarrow \exists s[\mathbb{S}ound] : \$(s)(a)$$

Grounding quantification over $\mathbb{S}ound$ results in:

$$\forall a[\mathbb{A}nimal] : makingSound(a) \Leftrightarrow \$(\widetilde{meow})(a) \lor \$(\widetilde{bark})(a).$$

Eliminating intensional terms results in:

$$\forall a[\mathbb{A}nimal] : makingSound(a) \Leftrightarrow meow(a) \lor bark(a).$$

The grounded formula is not well-typed as variable a of type $\mathbb{A}nimal$ occurs as an argument of type $\mathbb{C}at$ and $\mathbb{D}og$. Therefore we conclude that the initial formula is ill-typed.

Restoring the well-typedness of this formula necessitates guarding of the expression $\$(s)(a)$. Guarding this expression is challenging due to its intensional nature (i.e., variable s ranges over sounds). Consequently, the expression $\$(s)(a)$ has to be guarded depending on the value of s. This can be done by establishing a relation between *animal kinds* and their *specific sounds*. One common approach is introducing an auxiliary intensional type of *animal kinds* and intensional function mapping these *kinds to their sounds*. Type $\mathbb{K}ind$ (consisting of concepts $\widetilde{\mathbb{C}at}$ and $\widetilde{\mathbb{D}og}$) and function *soundOfKind* are declared as:

$$\text{type } \mathbb{K}ind <: \mathbb{C} := \{`(\mathbb{C}at), `(\mathbb{D}og)\} \qquad soundOfKind : \mathbb{K}ind \to \mathbb{S}ound$$

The following axioms define the mapping (the extension) of the function *soundOfKind*:

$$soundOfKind(`(\mathbb{C}at)) = `(meow) \qquad soundOfKind(`(\mathbb{D}og)) = `(bark)$$

Finally, the formula is guarded as:

$$\forall a[\mathbb{A}nimal] : makingSound(a) \Leftrightarrow \exists k[\mathbb{K}ind] : \$(k)(a) \land \$(soundOfKind(k))(a). \tag{5}$$

The grounded version of this formula corresponds to the formula (2), which is well-typed.

6 Guarded order-sorted intensional logic

Formula (5) enhances the original statement (2) by employing intensional constructs for guarding it. However, achieving this required the introduction of a helper function relating kinds to their sounds, despite this information being present in the type of predicates *meow* and *bark*. We address this issue by integrating guards (Section 4) and intensional logic (Section 5). First, we demonstrate it on the running example.

Example 7. *Recall the formula (2):*

$$\forall a[\mathbb{A}nimal] : makingSound(a) \Leftrightarrow (\mathbb{C}at(a) \wedge meow(a)) \vee (\mathbb{D}og(a) \wedge bark(a)).$$

Employing implicit guarding, the same can be expressed as:

$$\forall a[\mathbb{A}nimal] : makingSound(a) \Leftrightarrow [[meow(a)]] \vee [[bark(a)]].$$

Introducing quantification over $\mathbb{S}ound$ *(sounds) results in:*

$$\forall a[\mathbb{A}nimal] : makingSound(a) \Leftrightarrow \exists s[\mathbb{S}ound] : [[\$(s)(a)]].$$

In this example, we began with the explicitly guarded formula and condensed it into a compact version using implicit guarding and quantification over concepts. Consequently, the resulting statement is well-typed. Notably, variable a is implicitly constrained to the appropriate type based on the predicate to which it is applied. This reflects the main goal of the paper, which is incorporating subtyping polymorphism in order-sorted logic.

Example 8. *The same methodology applies to formula (3):*

$$\forall a[\mathbb{A}nimal] : \big((\mathbb{C}at(a) \Rightarrow meow(a)) \wedge (\mathbb{D}og(a) \Rightarrow bark(a))\big)$$

Using implicit guarding on this formula we obtain: $\forall a[\mathbb{A}nimal] : \langle\langle meow(a)\rangle\rangle \wedge \langle\langle bark(a)\rangle\rangle$*, and with quantifying over* $\mathbb{S}ound$*:* $\forall a[\mathbb{A}nimal] : \forall s[\mathbb{S}ound] : \langle\langle \$(s)(a)\rangle\rangle$.

Previous examples demonstrate principles for expressing properties of objects depending on their type using *guarded order-sorted intensional logic*. The following proposition generalizes the modeling principles discussed so far.

Proposition 2. *Given OSL vocabulary* Σ*:*

- *Let* $p_1, \dots p_m$ *be n-ary predicate symbols in* Σ
- *Let these symbols have type signature in* Σ *as:*

$$ts_\Sigma(p_1) = \mathbb{T}_{11} \times \dots \times \mathbb{T}_{1n} \to \mathbb{B} \qquad \dots \qquad ts_\Sigma(p_m) = \mathbb{T}_{m1} \times \dots \times \mathbb{T}_{mn} \to \mathbb{B}$$

- *Let* $\mathbb{S}_1, \dots, \mathbb{S}_n$ *be types in* Σ *such that:*

$$\mathbb{T}_{11} <: \mathbb{S}_1 \ \dots \ \mathbb{T}_{m1} <: \mathbb{S}_1 \qquad \dots \qquad \mathbb{T}_{1n} <: \mathbb{S}_n \ \dots \ \mathbb{T}_{mn} <: \mathbb{S}_n$$

- *Let* $\mathbb{P}$ *be a type in* Σ *such:* $\mathbb{P} <: \mathbb{C} := \{`(p_1), \dots, `(p_m)\}$.
- *Let* p *be a term of type* $\mathbb{P}$*, and* t_i *term of type* $\mathbb{S}_i$.

Then the following two expressions are well-typed:

$$[[\$(p)(t_1,\ldots,t_n)]] \qquad \langle\langle\$(p)(t_1,\ldots,t_n)\rangle\rangle$$

Proof. Term p denotes some $\widetilde{p_k}$ from $\mathbb{P}$ (recall that term '(p_k) stands for value $\widetilde{p_k}$). The symbol p_k is associated with a type term $\mathbb{T}_{k1}\times\cdots\times\mathbb{T}_{kn}\to\mathbb{B}$. Accordingly, $[[\$(p)(t_1,\ldots,t_n)]]$ stands for: $\mathbb{T}_{k1}(t_1)\wedge\cdots\wedge\mathbb{T}_{kn}(t_n)\wedge p_k(t_1,\ldots,t_n)$. Each of the terms t_i is of type $\mathbb{S}_i$ and hence also of type $\mathbb{U}$ (follows from Proposition 1 and Definition 3 rule (T-SUB)), so each atom $\mathbb{T}_{ki}(t_i)$ is well-typed (Definition 4). Finally, according to Definition 4 rule (G-C), atom $p_k(t_1,\ldots,t_n)$ is well-typed as the type of each t_i is $\mathbb{T}_{ki}$. The proof for $\langle\langle\$(p)(t_1,\ldots,t_n)\rangle\rangle$ is similar. □

Patterns characterized in this proposition are essential for expressing logical statements containing subtyping polymorphism as demonstrated in previous examples.

An important observation is that the presented approach enables the compact formalization of statements involving subtyping. For instance, formula (1) defining the predicate *makingSound* in native *OSL*, yields a formula whose length scales linearly with the number of animal kinds; i.e., the addition of another animal kind (e.g., mouse) would result in the formula growing in size. However, the logic presented in this paper is capable of expressing the same statements with formulae of constant length by utilizing the concepts introduced in Proposition 2, as demonstrated in the examples above. Formally:

Proposition 3. *Given the same environment as in Proposition 2, the following formulae cannot be expressed in an OSL formula with a length independent of the size of* $\mathbb{P}$*:*

$$\exists s[\mathbb{P}] : [[\$(s)(t_1,\ldots,t_n)]] \qquad \forall s[\mathbb{P}] : \langle\langle\$(s)(t_1,\ldots,t_n)\rangle\rangle$$

Proof. Rewriting these formulae into *OSL* requires the mentioning of all symbols in $\mathbb{P}$. □

7 Well-typedness in order-sorted intensional logic

We argued in Section 5 that the well-typedness of formulae with intensional language constructs is not trivial. In this section, we elaborate on these issues and propose the foundations for the typing system suitable for the new language.

Recall the methodology employed in formula (5) to guard the formula in Example 6. We introduced a function *soundOfKind* to establish the connection between animal kinds and their specific sounds. It is important to note that the well-typedness of formula (5) depends on the correct mapping of animal kinds to sounds by this function. For example, if the function incorrectly maps $\widetilde{\mathbb{C}at}$ to $\widetilde{bark}$, the formula (5) would be ill-typed. This underscores the dependence of well-typedness on the extensions (values) of types and functions. However, the typing system from Definition 3 cannot account for such dependencies, as the type of function *soundOfKind* does not provide sufficient information.

The first step towards a richer type system is the introduction of typing annotations that would clarify the typing of a concept. This idea is presented in [2, Section 4]. For example, when quantifying over concepts, one has to provide information about the type of these concepts.

$$\forall s \in \mathbb{C}[\mathbb{A}nimal \to \mathbb{B}] : \psi$$

In this statement variable s ranges over concepts from the vocabulary which are of type $\mathbb{A}nimal \to \mathbb{B}$. In Example 1 these are $\mathbb{C}at$, $\mathbb{D}og$, *bark*, *meow*. Similar information can be provided in the declaration of subtypes of concepts. For example, declaring a new type "kind of animals" (earlier introduced for fixing Example 6) requires annotating that each element of this type is a predicate over the "animal" type. Hence, the type "kind of animals" is a subtype of predicate concepts that are of type $\mathbb{A}nimal \to \mathbb{B}$.

$$\text{type } \mathbb{K}ind <: \mathbb{C}[\mathbb{A}nimal \to \mathbb{B}] := \{`(\mathbb{C}at), `(\mathbb{D}og)\}$$

However, this approach fails to fully support guarded *OSL*. For example, no type can substitute $(?)$ in the following declaration of type $\mathbb{S}ound$ from Example 5. This is because *meow* and *bark* are predicates over different types, $\mathbb{D}og$ and $\mathbb{C}at$ respectively.

$$\text{type } \mathbb{S}ound <: \mathbb{C}[(?) \to \mathbb{B}] := \{`(meow), `(bark)\}$$

Furthermore, essential for the well-typedness of formula (5) is the fact that objects of type $\mathbb{K}ind$ are type predicates, and hence can serve for guarding. To make this distinction, two new types can be added: $\mathbb{C}^T$ to represent type concepts and $\mathbb{C}^S$ for function/predicate concepts. We propose the following syntax for declaring $\mathbb{K}ind$ and $\mathbb{S}ound$:

$$\text{type } \mathbb{K}ind <: \mathbb{C}^T[\mathbb{A}nimal] := \{`(\mathbb{C}at), `(\mathbb{D}og)\}$$
$$\text{type } \mathbb{S}ound[t : \mathbb{K}ind] <: \mathbb{C}^S[t \to \mathbb{B}] := \{`(meow), `(bark)\}$$

Here, $\mathbb{C}^T[\mathbb{A}nimal]$ stands for type concepts that are subtypes of type $\mathbb{A}nimal$. Notation $\mathbb{S}ound[t : \mathbb{K}ind] <: \mathbb{C}^S[t \to \mathbb{B}]$ expresses that type $\mathbb{S}ound$ is subtype of predicate concepts of type $t \to \mathbb{B}$ where t is of type $\mathbb{K}ind$ (making $\mathbb{S}ound$ dependent on the value of t). Notice that this notation requires type checking for the declarations because types now have variables. In this example, it is necessary to show that variable t is of type $\mathbb{C}^T$.

Finally, the function *soundOfKind* can be declared in the following way:

$$soundOfKind : k \to \mathbb{S}ound[k] \mid k : \mathbb{K}ind$$

Here, notation $\mathbb{S}ound[k]$ expresses the projection of type $\mathbb{S}ound$ to only these predicates that are over type k. This is essential for forming the connection between the types of domain and the range of the function. Informally, this declaration aims to express that function *soundOfKind* maps kinds $\mathbb{K}ind$ to the sounds of that kind $\mathbb{S}ound[k]$. In particular, based on the type information, sounds of kind $\mathbb{S}ound[k]$ can be any predicate with the typing signature $k \to \mathbb{B}$. Using this information, it is possible to conclude that formula (5) is well-typed. In particular, given that variable a is of type $\mathbb{A}nimal$ and k of type $\mathbb{K}ind$ the following reasoning can be employed to make a judgment $(\$(k)(a) \wedge \$(soundOfKind(k))(a)) : \mathbb{B}$ (which is the challenging part of formula (5)):

- $\$(k)(a)$ is well-typed as k is some type symbol that is subtype of $\mathbb{A}nimal$ and a is of type $\mathbb{A}nimal$, and per Definition 4 types can appear as predicates.
- Since k is a type-symbol, $\$(k)(a)$ can be used for guarding the other part of the conjunction (similar to the (G-C) rule from Definition 4).
- $\$(soundOfKind(k))(a)$ is well-typed because: (*i*) $soundOfKind(k)$ is of type $k \to \mathbb{B}$ (*ii*) Variable a is of type k thanks to the guard $\$(k)(a)$ (*iii*) $\$(soundOfKind(k))(a)$ is of type $\mathbb{B}$ as variable a (of type k) is applied to some predicate of type $k \to \mathbb{B}$.

To a certain extent, the typing system illustrated in this section resembles the idea of dependent types [8, Chapter 6, Section 30.5]. In type theory, a type is considered dependent if its definition relies on a value. For example, a function that adds a new number to a list takes a number and a list of length n as arguments and returns a list of length $n+1$. Similarly, the function *soundOfKind* takes the intension of a type (a subtype of $\mathbb{A}nimal$) as an argument and returns the intension of a unary predicate over that type. Due to its extensiveness, formalizing such a typing system for order-sorted intensional logic and investigating its relation to dependent types remains within the scope of future work.

8 Semantics of the language

The formal model semantics of the logic presented in this paper rely on a combination of order-sorted logic [1, Section 4.2] and intensional logic [2, Section 3.2]. Note that in all our examples, extensions of types and functions ranging over concepts are fixed (i.e., $\mathbb{S}$*ound* contains exactly *bark* and *meow*). This allows for grounding intensional language constructs and semantically reducing the logic to standard *OSL*. However, this section outlines the semantics of the order-sorted intensional logic. First, we define the notion of structure, a value assignment to vocabulary symbols.

Definition 7. *A structure $\mathfrak{A}$ over an OSL vocabulary Σ interprets all symbols s in Σ (denoted as $s^{\mathfrak{A}}$) such that:*

1. *The value of each type symbol $\mathbb{T}$ in T_Σ is a non-empty set $\mathbb{T}^{\mathfrak{A}}$.*
   - *Type $\mathbb{B}$ (boolean) is always assigned the set of truth values $\mathbb{B}^{\mathfrak{A}} = \{\mathfrak{true}, \mathfrak{false}\}$*
   - *Type $\mathbb{N}$ (natural numbers) is always assigned the set $\mathbb{N}^{\mathfrak{A}} = \{0,1,2,\dots\}$*
   - *Type $\mathbb{C}$ (concepts) is assigned the set $\mathbb{C}^{\mathfrak{A}} = \{\widetilde{s} \mid s \in S_\Sigma \cup T_\Sigma\}$. Here $\widetilde{s}$ is the atomic object formally representing the concept behind the symbol s.*
2. *If type symbol $\mathbb{T}$ is a direct subtype ($<:_\Sigma$) of $\mathbb{T}_1$, then $\mathbb{T}^{\mathfrak{A}} \subseteq \mathbb{T}_1^{\mathfrak{A}}$.*
3. *Each symbol s in S_Σ with type signature $ts_\Sigma(s) = (\mathbb{T}_1 \times \cdots \times \mathbb{T}_n) \to \mathbb{T}$, is assigned a set $s^{\mathfrak{A}} \subseteq \mathbb{T}_1^{\mathfrak{A}} \times \cdots \times \mathbb{T}_n^{\mathfrak{A}} \times \mathbb{T}^{\mathfrak{A}}$ such that:*
   - *for each tuple $(d_1,\dots,d_n) \in \mathbb{T}_1^{\mathfrak{A}} \times \cdots \times \mathbb{T}_n^{\mathfrak{A}}$ there is an element $e \in \mathbb{T}^{\mathfrak{A}}$ such that $(d_1,\dots,d_n,e_1) \in s^{\mathfrak{A}}$.*
   - *for all tuples $(d_1,\dots,d_n,e_1),(d_1,\dots,d_n,e_2) \in s^{\mathfrak{A}}$, it holds that $e_1 = e_2$.*
   - *If s is a type predicate $\mathbb{T}$, then $s^{\mathfrak{A}} = \{(d,\mathfrak{true}) \mid d \in \mathbb{T}\} \cup \{(d,\mathfrak{false}) \mid d \in \mathbb{U} \setminus \mathbb{T}\}$.*

   If s is a function symbol and $(d_1,\dots,d_n,e) \in s^{\mathfrak{A}}$, we write that $s^{\mathfrak{A}}(d_1,\dots,d_n) = e$.

A common assumption is that each domain object has an identifier, a symbol that makes it possible to directly refer to that value from the theory. With the notion of a structure formalized, we proceed with defining the value of an expression in a structure.

Definition 8. *Given vocabulary Σ, let α be an OSL expression (over Σ), and $\mathfrak{A}$ a structure interpreting all symbols in Σ. Further, let, for each free variable x occurring in α as an argument of type $\mathbb{T}$, structure $\mathfrak{A}$ assign value $x^{\mathfrak{A}} \in \mathbb{T}^{\mathfrak{A}}$ (with $\mathfrak{A}[x:d]$ we denote that structure $\mathfrak{A}$ is extended with assignment of value d to variable x). The* ***value of α in $\mathfrak{A}$****, denoted as $[\![\alpha]\!]^{\mathfrak{A}}$, is defined by induction on the structure of α:*

$$[\![x]\!]^{\mathfrak{A}} = x^{\mathfrak{A}} \qquad [\![\textit{\textbf{true}}]\!]^{\mathfrak{A}} = \mathfrak{true} \qquad [\![\textit{\textbf{false}}]\!]^{\mathfrak{A}} = \mathfrak{false}$$

$$[\![f(\tau_1,\dots,\tau_n)]\!]^{\mathfrak{A}} = f^{\mathfrak{A}}([\![\tau_1]\!]^{\mathfrak{A}},\dots,[\![\tau_n]\!]^{\mathfrak{A}}) \qquad [\![p(\tau_1,\dots,\tau_n)]\!]^{\mathfrak{A}} = p^{\mathfrak{A}}([\![\tau_1]\!]^{\mathfrak{A}},\dots,[\![\tau_n]\!]^{\mathfrak{A}})$$

$$[\![\neg\phi]\!]^{\mathfrak{A}} = \begin{cases} \mathfrak{true}, & \textit{if } [\![\phi]\!]^{\mathfrak{A}} = \mathfrak{false}; \\ \mathfrak{false}, & \textit{if } [\![\phi]\!]^{\mathfrak{A}} = \mathfrak{true}; \end{cases}$$

$$[\![\phi_1 \vee \phi_2]\!]^{\mathfrak{A}} = \begin{cases} \mathfrak{true}, & \textit{if } [\![\phi_1]\!]^{\mathfrak{A}} = \mathfrak{true} \textit{ or } [\![\phi_2]\!]^{\mathfrak{A}} = \mathfrak{true}; \\ \mathfrak{false}, & \textit{if } [\![\phi_1]\!]^{\mathfrak{A}} = [\![\phi_2]\!]^{\mathfrak{A}} = \mathfrak{false}; \end{cases}$$

$$[\![\exists x[\mathbb{T}] : \phi]\!]^{\mathfrak{A}} = \begin{cases} \mathfrak{true}, & \textit{if for some } d \in \mathbb{T}^{\mathfrak{A}}, [\![\phi]\!]^{\mathfrak{A}[x:d]} = \mathfrak{true}; \\ \mathfrak{false}, & \textit{if for all } d \in \mathbb{T}^{\mathfrak{A}}, [\![\phi]\!]^{\mathfrak{A}[x:d]} = \mathfrak{false}; \end{cases}$$

$$[\![`(s)]\!]^{\mathfrak{A}} = \widetilde{s}, \quad \textit{for } s \in S_\Sigma \cup T_\Sigma$$

$$[\![\$(\tau)(\tau_1,\ldots,\tau_n)]\!]^{\mathfrak{A}} = s^{\mathfrak{A}}([\![\tau_1]\!]^{\mathfrak{A}},\ldots,[\![\tau_n]\!]^{\mathfrak{A}}),\ \textit{for } s \in S_\Sigma \cup T_\Sigma \textit{ such that } [\![\tau]\!]^{\mathfrak{A}} = \widetilde{s}$$

An OSL sentence ψ over vocabulary Σ is ***satisfied*** *in a structure $\mathfrak{A}$ (over Σ), denote as $\mathfrak{A} \models \psi$, if and only if* $[\![\psi]\!]^{\mathfrak{A}} = \mathfrak{true}$.

As discussed in Section 5, not every order-sorted intensional expression is meaningful (as illustrated by the example: $bark(tom)$). Consequently, attempting to define the value of such expressions is not meaningful. These expressions can be excluded by enhancing the typing system as explained in Section 7.

9 Related work and discussion

Frame Logic (F-logic), introduced in [5], is a knowledge representation language that combines conceptual modeling with object-oriented and frame-based languages. In this language, it is possible to use types as predicates which is sufficient for expressing formulae like (2) and (3). Logic programming incorporating polymorphically order-sorted types is investigated in [9]. The Flora-2 [6] system combines F-logic and HiLog [3], resulting in an even more expressive language. The key differences between these languages and guarded *OSL* are: *(i)* F-logic is mainly utilizing subtyping from the perspective of object-oriented paradigm while the focus of this paper is on a more general notion of types. *(ii)* Results of these papers are related to *parametric polymorphism* [8, Chapter 23] rather than subtyping polymorphism. *Parametric (Ad hoc) polymorphism* includes generic types, polymorphic predicate and function symbols and quantification over types. An example is *mother* and *father* functions, mapping animals of a certain kind to another animal of that same kind. Using parametric polymorphism the typing signature of this function can be expressed as ($mother : \forall k <:_\Sigma \mathbb{A}nimal\ .\ k \to k$). Even though this notation strongly resembles the idea presented in Section 7 they are different. Here, variable k ranges over types, while in the other example, this does not have to be the case. However, the dependent type approach with intensional logic can sometimes simulate parametric polymorphism. In this particular example: ($mother : k \to k \mid k : \mathbb{C}^T[\mathbb{A}nimal]$). *(iii)* These languages lack intensional aspects. While HiLog allows for higher-order language constructs, it does not include concepts. This means that using functions such as *soundOfKind* to "compose" formulae is not possible. In other words, one can see the intensional logic presented in this paper as a mechanism for expressing templates of formulae. This is because objects from the vocabulary are first-class citizens. This is not the case with the higher-order logic. *(iv)* Implicit type guarding is not supported in these languages. In particular, to the best of our knowledge, no other languages use such language constructs (except for our previous work [7] where guards ensure the safe application of partial functions). However, this paper demonstrates the importance of implicit guarding and power coming from combining it with intensional logic.

The points *(iii)* and *(iv)* suggest that these languages may encounter similar problems to those concerning *OSL* discussed in Section 3 and intensional logic from Section 5.

On the other side, the scope of this paper is limited to subtyping polymorphism. Future research should explore how the approach presented in this paper relates to *parametric (Ad hoc) polymorphism*. In particular, it is worth investigating whether the two typing systems have the same expressive power. Another research question that opens here is what if we perceive typed logic as a logic of partial predicates, what is then the relation between guarding presented in this paper and guarding that ensures arguments of a function are in its domain of definedness (our previous work [7]).

Similar to the approach demonstrated in formula (5), it is possible to define higher-order functions in HiLog to map propositions to propositions, thereby achieving similar outcomes. However, this approach carries the same issues as the one with intensional logic. Namely, it requires introduction of new functions and predicates representing the typing relation between different concepts which is redundant as this information is present in the typing signature of these concepts. This issue was discussed in Section 6. Similar issues apply to many imperative programming languages, such as Python, where *dynamic function invocation* can yield similar results but with the price of introducing redundant type information. Dynamic function invocation allows one to store names of functions in variables and then invoke these functions by using the variable.

In conclusion, many declarative (logic-based) and imperative languages can achieve similar results as presented in this paper. However, mainly due to the lack of implicit guarding and intensional aspects of the language, these languages do not support the subtyping discussed in this paper as a native language construct. To the best of our knowledge, there are no such knowledge representation languages.

10 Conclusion

In this paper, we addressed the challenge of subtyping polymorphism within order-sorted logic. Through our investigation, we identified two essential requirements: intensional logic and implicit guarding with typing assertions. Consequently, we introduced *guarded order-sorted intensional logic* and demonstrated its effectiveness for this task.

The main contributions of this paper are: *(i)* implicit guarding, language constructs introduced in Definition 5 allowing conditioning of types for terms based on their application; *(ii)* combining implicit guarding and intensional logic (i.e., quantification over concepts) for expressing subtyping polymorphism, as elaborated in Propositions 2 and 3. Additionally, this paper opens two new research topics: the well-typedness conditions of *guarded order-sorted intensional logic* and its relation to dependent types (see Section 7), and second, the relation of order-sorted logic as presented in this work and logic of partial functions (see Section 9).

Acknowledgments

Special thanks to Maurice Bruynooghe for his thorough reviews of this paper. Thanks to Robbe Van den Eede and Linde Vanbesien for valuable discussions. Thanks to Tobias Reinhard and Justus Fasse for their insightful reviews of the early versions of this paper.

References

[1] Christoph Beierle, Ulrich Hedtstück, Udo Pletat, Peter H. Schmitt & Jörg H. Siekmann (1992): *An Order-Sorted Logic for Knowledge Representation Systems*. Artif. Intell. 55(2), pp. 149–191, doi:10.1016/0004-3702(92)90055-3.

[2] Pierre Carbonnelle, Matthias van der Hallen & Marc Denecker (2023): *Quantification and aggregation over concepts of the ontology*. In Enrico Pontelli, Stefania Costantini, Carmine Dodaro, Sarah Alice Gaggl, Roberta Calegari, Artur S. d'Avila Garcez, Francesco Fabiano, Alessandra Mileo, Alessandra Russo & Francesca Toni, editors: Proceedings 39th International Conference on Logic Programming, ICLP 2023, Imperial College London, UK, 9th July 2023 - 15th July 2023, EPTCS 385, pp. 213–226, doi:10.4204/EPTCS.385.22.

[3] Weidong Chen, Michael Kifer & David Scott Warren (1993): *HILOG: A Foundation for Higher-Order Logic Programming*. J. Log. Program. 15(3), pp. 187–230, doi:10.1016/0743-1066(93)90039-J.

[4] Melvin Fitting (2004): *First-order intensional logic*. *Ann. Pure Appl. Log.* 127(1-3), pp. 171–193, doi:10.1016/J.APAL.2003.11.014.

[5] Michael Kifer & Georg Lausen (1989): *F-Logic: A Higher-Order language for Reasoning about Objects, Inheritance, and Scheme*. In James Clifford, Bruce G. Lindsay & David Maier, editors: *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, USA, May 31 - June 2, 1989*, ACM Press, pp. 134–146, doi:10.1145/67544.66939.

[6] Michael Kifer, Georg Lausen & James Wu (1995): *Logical Foundations of Object-Oriented and Frame-Based Languages*. *J. ACM* 42(4), pp. 741–843, doi:10.1145/210332.210335.

[7] Djordje Markovic, Maurice Bruynooghe & Marc Denecker (2023): *Towards Systematic Treatment of Partial Functions in Knowledge Representation*. In Sarah Alice Gaggl, Maria Vanina Martinez & Magdalena Ortiz, editors: *Logics in Artificial Intelligence - 18th European Conference, JELIA 2023, Dresden, Germany, September 20-22, 2023, Proceedings*, *Lecture Notes in Computer Science* 14281, Springer, pp. 756–770, doi:10.1007/978-3-031-43619-2_51.

[8] Benjamin C Pierce (2002): *Types and programming languages*. MIT press.

[9] Gert Smolka (1988): *Logic Programming with Polymorphically Order-Sorted Types*. In Jan Grabowski, Pierre Lescanne & Wolfgang Wechler, editors: *Algebraic and Logic Programming, International Workshop, Gaussig, GDR, November 14-18, 1988, Proceedings*, *Lecture Notes in Computer Science* 343, Springer, pp. 53–70, doi:10.1007/3-540-50667-5_58.

[10] Hao Wang (1952): *Logic of Many-Sorted Theories*. *J. Symb. Log.* 17(2), pp. 105–116, doi:10.2307/2266241.

A Coq Formalization of Unification Modulo Exclusive-Or

Yichi Xu
Worcester Polytechnic Institute
Massachusetts, USA
yxu10@wpi.edu

Daniel J. Dougherty
Worcester Polytechnic Institute
Massachusetts, USA
dd@wpi.edu

Rose Bohrer
Worcester Polytechnic Institute
Massachusetts, USA
rbohrer@wpi.edu

Equational Unification is a critical problem in many areas such as automated theorem proving and security protocol analysis. In this paper, we focus on XOR-Unification, that is, unification modulo the theory of exclusive-or. This theory contains an operator with the properties Associativity, Commutativity, Nilpotency, and the presence of an identity. In the proof assistant Coq, we implement an algorithm that solves XOR unification problems, whose design was inspired by Liu and Lynch, and prove it sound, complete, and terminating. Using Coq's code extraction capability we obtain an implementation in the programming language OCaml.

1 Introduction

Unification is a fundamental concept used across various domains such as logic programming, type systems, and constraint solving. In logic programming, it enables pattern matching and logical inference, crucial for problem-solving in languages like Prolog [7]. Within type systems, such as those in Haskell and Scala [20], unification supports type inference, allowing compilers to ensure type safety and catch type errors at compile-time rather than runtime. Additionally, in constraint solving, it helps manage and resolve variable constraints, essential for applications in scheduling and planning. This work, however, is motivated by applications of unification to security protocol analysis. A common way to analyze protocols is to perform syntactic unification with the protocol rules to explore some space of reachable states. If an "attack" state is reachable from the initial state then an attack exists and the protocol is flawed.

However, the limitation of using syntactic unification to analyze protocols is that it only captures the case when terms, representing messages, can be made exactly the same, which in many protocols is not enough. For example, the Vernam cipher and cipher-block chaining mode for block ciphers rely on exclusive-or (XOR) [16]. There exists protocols which seem secured if XOR is left uninterpreted, but whose flaws are revealed when XOR is an interpreted operator. For example, the original version of Bull's recursive authentication protocol was formally proved correct in the Dolev-Yao model, but this XOR-based protocol was vulnerable to an attack that exploited the self-cancellation property [24]. XOR Unification is important because it enables a more accurate analysis of XOR-based protocols. Because unification is a key ingredient of logic programming and because logic programming has established applications to security analysis [30], we believe research about XOR Unification may help enable logic programming-based analyses of XOR-based protocols in the long term.

In this paper, we adopt a modified version of the algorithm developed by Liu and Lynch [12], then implement it and prove it correct in Coq. This work is important because it increases our confidence in Liu and Lynch's work. The primary reason for not adopting the full algorithm was time constraints. Consequently, we decided to exclude uninterpreted functions and homomorphic function from our implementation. This decision was based on the understanding that both uninterpreted and homomorphic functions can be treated as standard terms under specific constraints. For uninterpreted functions, we can

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 267–273, doi:10.4204/EPTCS.416.23

compare their function symbols and conduct syntactic unification within the functions. For homomorphic functions, we can reduce them to their normal form before proceeding with syntactic comparison.

2 Related Work

Syntactic unification is unification modulo the empty equational theory. There are many algorithms for syntactic unification, but there are only a few which have been verified and formalized. The earliest formalization is the algorithm from Manna and Waldinger [13], which was proved by Paulson [22] using LCF (Logic for Computable Functions). This formalization is used as a basis for later research by Sternagel and Thiemann [25] in Isabelle. Urban, Pitts, and Gabbay [28] also formalized first-order unification in Isabelle. A relatively recent formalization for syntactic unification is from Avelar, Galdino, deMoura, and Ayala-Rincon [1] using PVS (Prototype Verification System).

E-unification is unification modulo an equational theory. Dougherty [8] has verified two algorithms for Boolean unification. Ayala-Rincón *et. al.* [2] have verified an AC(Associativity and Commutativity)-Unification algorithm using PVS. For XOR unification, there are only a few algorithms and no formalization. Tuengethal, Kusters and Turuani [27] mentioned a relatively easy and intuitive way to design such an algorithm by combining theories such that their overall output satisfies the XOR properties. Guo, Narendran, and Wolfram [10] mentioned using Gaussian elimination over a Boolean ring to compute unifiers for XOR unification. Liu and Lynch [12] give several terminating inference rules to solve XOR unification. However, the above papers only give algorithms but not a formalization. Therefore in this paper, we decided to do a formalization over their work in Coq so we can be more confidence in the algorithm.

3 Background

The XOR operator is common operator seen in many protocols; a famous example is the Vernam Cipher[17, Definition.1.39]:

$$c_i = m_i \oplus k_i, 1 \leq i \leq t. \tag{1}$$

where t is the length of the message in digits, i ranges over the digits of the message, c_i is the digit's ciphertext, m_i the message, and k_i the key. That is, the Vernam cipher XOR's a message with a key of the same length. Such a cipher is decrypted by applying the XOR operation a second time with the same key.

We formally state the axioms for XOR, where the signature is $\Sigma = \{\oplus, 0\}$:

- Associativity: $x \oplus (y \oplus z) = (x \oplus y) \oplus z$
- Commutativity: $x \oplus y = y \oplus x$
- Unity: $x \oplus 0 = x$
- Nilpotency: $x \oplus x = 0$

Then, we present the modified rewrite system that is used to compute the unifiers. This rewrite system *amounts to an algorithm* for XOR Unification, i.e., the algorithm is to apply the first applicable rule, terminating when none applies. In this paper, we use two sets $\Gamma \| \Lambda$ to keep track of the computation progress: Γ denotes the unification problem consisting of a set of equations $\{S \approx^?_E 0\}$, where S is any term, $\approx^?_E$ is the symbol for deciding two terms on both side are the same or not under equational theory E, and 0 is the unit term. Because of the Nilpotency Axiom, we can move the term from the right-hand side to the left-hand side without losing equivalency; i.e. $t_1 \approx_E^? t_2 \to t_1 \oplus t_2 \approx_E^? t_2 \oplus t_2$ and $t_2 \oplus t_2 \approx_E 0$. And Λ denotes a set of equations in solved form. Initially, the unification problems are stored in Γ, while Λ remains empty. If a system is in normal form regarding these inference rules, then Λ is in solved form if the original problem is solvable.

We introduce the two inference rules used in this algorithm:

Trivial: seeks a problem that is already solved and deletes it

$$\frac{\Gamma \cup \{0 \approx_E^? 0\} \| \Lambda}{\Gamma \| \Lambda} \tag{2}$$

Variable Substitution: seeks a solved form and applies this substitution to the whole system

$$\frac{\Gamma \cup \{x \oplus S \approx_E^? 0\} \| \Lambda}{\sigma\Gamma \| \sigma\Lambda \cup \{x \approx_E^? S\}} \tag{3}$$

where $\sigma = x \mapsto S$ and $x \notin S$, i.e. the occurs check passes.

In the Coq development, we need to prove this set of inference rules correct. Correct here means it will return an idempotent mgu (most general unifier) of the original problem if it is solvable, and this rewrite system will terminate for all input, see the formal theorem stated in Figure 4.

4 Coq Implementation

This section illustrates the definition of different data structures, the algorithm, and the theorems in Coq. Please note that we only provide the statement of the theorems in this section, as the full proofs have 11,000 lines of code. For the complete development, please refer to our Coq code [29]. For an introduction to Coq notation, see background material [26].

4.1 Basic Data structure

Given that this work only concerns constants, variables, and the XOR operator ($\oplus$), abstract syntax of formulas can be described in the following way in Coq.

```
Definition var := string.
Inductive term: Type :=
| C  :  nat -> term #Constant
| V  :  var -> term #Variable
| Oplus  : term -> term -> term.
Definition T0 :term:= C 0. # Short for Constant 0, unit in unity axiom
Notation "x +' y" := (Oplus x y) (at level 50, left associativity).
```

Figure 1: Term Definition

The constructor `C` takes a natural number, which is a built in data structure from Coq, as its input and outputs a constant term, while the constructor `V` takes a string as input and outputs a variable term. The operator $\oplus$ takes two terms as inputs and outputs a nested $\oplus$ term. Note that constant `T0` is the unit of XOR.

After introducing the fundamental term representations in Coq, it is necessary to define the equivalence relation modulo XOR (shown in Figure 2). In addition to the four axioms of associativity, commutativity, unity, and nilpotency, this relation must also satisfy the properties of reflexivity, symmetry, and transitivity, as it is an equivalence relation. Since this is a congruence relation, we also must define a compatibility axiom oplus compat.

```
Reserved Notation "x == y" (at level 70).
Inductive eqv : term -> term -> Prop :=
| eqvA: forall x y z,    (x +’ y) +’ z == x +’ (y +’ z)
| eqvC: forall x y, x +’ y ==  y +’ x
| eqvU: forall x,  T0 +’ x  == x
| eqvN: forall x,  x +’ x  == T0
| eqv_ref: forall x,  x == x
| eqv_sym: forall x y,  x == y ->  y == x
| eqv_trans: forall x y z,  x == y ->  y == z ->  x == z
| Oplus_compat : forall x x’ ,  x == x’ -> forall y y’ ,
      y == y’ -> x +’ y ==  x’ +’ y’
where "x == y" := (eqv x y).
```

Figure 2: Equivalence Definition

4.2 XOR-Rewrite System

Processing proofs with nested terms is not a simple job, consider the following two terms:

$$z \oplus a \oplus (b \oplus c) \oplus a \oplus (b \oplus c) \oplus z \tag{4}$$

$$d \oplus (a \oplus e) \oplus ((b \oplus (d \oplus e)) \oplus c) \oplus a \oplus (b \oplus c) \tag{5}$$

Both terms can ultimately be reduced to zero; however, accomplishing this reduction is challenging with nested forms, and the complexity is further increased when utilizing the Coq notation described earlier.

Consequently, an alternative approach was adopted. We reduce each term into a list-shaped normal form. More specifically, we designed two functions: $f_{tlt}()$ and $f_{ltt}()$, to transform a term into a list and back. We use the name `lTerm` to describe this representation, and a predicate $\approx\approx$ for equivalency between `lTerm`. Then, we need to prove that these two predicates capture the same equivalence for these two data representations. We state our lemmas below:

Lemma 1. $\forall (t1, t2 : term), t1 \approx_{XOR} t2 \leftrightarrow f_{tlt}(t1) \approx\approx f_{tlt}(t2)$

Lemma 2. $\forall (tl1, tl2 : lTerm), tl1 \approx\approx tl2 \leftrightarrow f_{ltt}(tl1) \approx_{XOR} f_{ltt}(tl2)$

Then we designed a rewrite system $f_R()$ such that equivalent `lTerms` are syntactically equal after rewriting. In other words, the following theorem must hold true:

Theorem 1. $\forall (tl1, tl2 : lTerm), tl1 \approx\approx tl2 \leftrightarrow f_R(tl1) = f_R(tl2)$

Once we have the lemmas and theorem in place, we can use the syntactic checker, which checks that both sides are identical, to compute the results for XOR equivalence. Note that the other benefit of this approach to prove the correctness of unification is that this can be easily modified to include homomorphic functions and uninterpreted functions in the future. The reason why we developed this representation is because it allowed us to normalize terms and minimize their syntactic complexity which makes the proofs easier.

4.3 XOR-Unification Algorithm and Correctness

To set up the final algorithm, we first need to convert the raw input problems to reduced problems (`lTerm` form), and then perform a fixed number of steps on the reduced problem. If the left-hand side of the problem set is empty after these steps, then the problem is solvable, and the function returns the right-hand side, which is the solved form of the original problem, and then transforms it into a substitution. If the left-hand side is not empty, it means the problem is not solvable and the function returns `None`.

Here are properties we proved in Coq for correctness: If the algorithm returns `None` then original unification problem is not solvable. If the algorithm returns some substitution, then this substitution

```
Theorem XORUnification_solves:forall(ps:problems)(sb:sub),
  XORUnification ps = Some sb -> solves_problems sb ps.

Theorem XORUnification_mgu:forall(ps:problems)(sb:sub),
  XORUnification ps = Some sb -> mgu_xor sb ps.

Theorem XORUnification_idpt:forall(ps:problems)(sb:sub),
  XORUnification ps = Some sb -> idempotent sb.
```

Figure 3: Solution found imply solution, solves, most general and idempotent

solves the original unification problem. Specifically, the substitution is the most-general unifier (mgu) of the problem and is idempotent.

We also proved that if the problem does not have a solution, then the algorithm will return `None`.

```
Definition problems_unifiable(ps:problems):Prop:=
  exists sb:sub, solves_problems sb ps.

Theorem unifiable_return_sub:forall(ps:problems),
  problems_unifiable ps -> (exists sb:sub, XORUnification ps = Some sb).

Theorem not_unifiable_return_None:forall(ps:problems),
  ~(problems_unifiable ps) -> XORUnification ps = None.
```

Figure 4: Not solvable implies no solution found

To sum up, in this development, we proved that: If the original unification problem is solvable, then the algorithm will return a substitution that is a most general unifier and is idempotent. If the original unification problem is not solvable then the algorithm will return `None`.

The Project took 1 person-year. This includes time spent learning Coq and iterating on intermediate proof attempts. This process included learning the basics of term rewriting, deciding project scope, implementation, and revision. After 1 person-year of work, the Coq implementation has roughly 11,000 lines of code. Since most of the proofs are not automated, the complete proof can be checked in Coq in less than 1 second.

In the end, the difficulty of proving soundness and completeness was similar because they rely on the same supporting lemmas. The reason why we chose the approach of reducing terms to `lTerms` is because we tried to prove things with nested terms at beginning, but comparing equivalency between two terms is a major challenge as illustrated in Figures 4 and 5. This project involved constantly substituting in new terms and comparing two terms to see whether they are solved or whether the equation is balanced. The lack of an efficient decision procedure for equivalence becomes a substantial problem during some specific proofs involving substitution, motivating our approach. This helped with processing complicated proofs relating to nested terms, because substitution takes place in `lTerm` and comparing two terms consists of checking whether their reduced forms are syntactically the same. This syntactic check is easier on lists. Moreover, we believe lots of other equational theories can adopt similar approaches for reasoning about equivalence.

Some problems we took a long time to prove are surprisingly not related to the inference rules or their correctness. Most are about reducing terms to `lTerms`. One key challenge was building a library for showing intuitive properties about the syntax tree, such as proving that different constructors yield disequal terms. For example, `C` 1 is not equivalent to `V` ”x”, and `C` 1 is not equivalent to `C` 2.

5 Conclusion and Future Work

We highlight two areas for future work. First, in the short term, we propose to extend our Coq formalization with uninterpreted and homomorphic functions. Second, in the long term, we propose that our formalization of XOR Unification can be used as the basis of a formalized implementation of logic programmnig with XOR, useful as a tool for security protocol analysis.

Supporting uninterpreted and homomorphic functions would require extensions to our data structures, rules, and proofs. The data structure changes would be only a few lines of code, and the rules would not be much longer. However, the proofs would be complicated significantly, as they would need to keep track of sets of uninterpreted and homomorphic functions throughout. Moreover, the addition of uninterpreted function symbols causes the unification algorithm to become non-deterministic. The addition of non-determinism increases the conceptual difficulty of the proofs.

In the long term, automated analysis of security exploits using logic programming with XOR is an exciting potential application of this work. Searching for security exploits in general is an established application of logic programming [30]. This suggests logic programming-based approaches may also be worth exploring when analyzing security protocols specifically. In order to express many security protocols [17] as logic programs, one must support an XOR operator and thus XOR unification.

Security tools are worthy of the strongest possible correctness guarantees, an observation which highlights the potential impact of our work. Our work provides the first formalization of XOR unification with a machine-checked proof from which guaranteed-correct code can be extracted. In so doing, we provide a strong foundation for any high-stakes analysis using logic programming with XOR.

References

[1] Andréia Borges Avelar, André Luiz Galdino, Flávio Leonardo Cavalcanti de Moura & Mauricio Ayala-Rincón (2014): *First-order unification in the PVS proof assistant*. Logic Journal of the IGPL 22(5), pp. 758–789, doi:10.1093/jigpal/jzu012.

[2] Mauricio Ayala-Rincón, Maribel Fernández, Gabriel Ferreira Silva & Daniele Nantes Sobrinho (2022): *A Certified Algorithm for AC-Unification*. In Amy P. Felty, editor: 7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel, LIPIcs 228, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 8:1–8:21, doi:10.4230/LIPICS.FSCD.2022.8.

[3] Franz Baader (1991): *Unification in a Theory of Primitive Recursive Functions with Applications to Semigroups and Groups*. Information and Computation 95(1), pp. 1–36.

[4] Franz Baader & Tobias Nipkow (1998): *Term Rewriting and All That*. Cambridge University Press, doi:10.1017/CBO9781139172752.

[5] Yves Bertot & Pierre Castéran (2013): *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media.

[6] Jacques Fleuriot Carlin MacKenzie, James Vaughan: *Archive of formal proofs*. Available at https://www.isa-afp.org/. Accessed: 2023-01-11.

[7] William F. Clocksin & Christopher S. Mellish (2003): *Programming in Prolog: Using the ISO Standard*. Springer Science & Business Media, doi:10.1007/978-3-642-55481-0.

[8] Daniel J Dougherty (2019): *A Coq Formalization of Boolean Unification*.

[9] H. Geuvers (2009): *Proof assistants: History, ideas and future*. Sadhana 34(1), p. 3–25, doi:10.1007/s12046-009-0001-5.

[10] Qing Guo, Paliath Narendran & David A Wolfram (1996): *Unification and matching modulo nilpotence*. In: International Conference on Automated Deduction, Springer, pp. 261–274, doi:10.1007/3-540-61511-3_90.

[11] Mauro Jaskelioff & Stephan Merz (2005): *Proving the Correctness of Disk Paxos*. https://isa-afp.org/entries/DiskPaxos.html, Formal proof development.

[12] Zhiqiang Liu & Christopher Lynch (2011): *Efficient general unification for XOR with homomorphism*. In: International Conference on Automated Deduction, Springer, pp. 407–421, doi:10.1007/978-3-642-22438-6_31.

[13] Zohar Manna & Richard Waldinger (1983): *Deductive synthesis of the unification algorithm*. In: Computer Program Synthesis Methodologies, Springer, pp. 251–307, doi:10.1007/978-94-009-7019-9_8.

[14] Alberto Martelli & Ugo Montanari (1982): *An Efficient Unification Algorithm*. ACM Transactions on Programming Languages and Systems 4(2), pp. 258–282, doi:10.1145/357162.357169.

[15] Matthew Maurer (2018): *Holmes: Binary analysis integration through datalog*. Ph.D. thesis, Carnegie Mellon University.

[16] Alfred J Menezes, Paul C Van Oorschot & Scott A Vanstone (2018): *Handbook of applied cryptography*. CRC press, doi:10.1201/9781439821916.

[17] Alfred J. Menezes, Scott A. Vanstone & Paul C. Van Oorschot (1996): *Handbook of Applied Cryptography*, 1st edition. CRC Press, Inc., USA.

[18] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn & Jakob von Raumer (2015): *The Lean theorem prover (system description)*. In: International Conference on Automated Deduction, Springer, pp. 378–388, doi:10.1007/978-3-319-21401-6_26.

[19] Tobias Nipkow, Lawrence C Paulson & Markus Wenzel (2002): *Isabelle/HOL: a proof assistant for higher-order logic*. 2283, Springer Science & Business Media, doi:10.1007/3-540-45949-9_6.

[20] Martin Odersky, Lex Spoon & Bill Venners (2016): *Programming in Scala*, 3 edition. Artima Press.

[21] Sam Owre, John M Rushby & Natarajan Shankar (1992): *PVS: A prototype verification system*. In: International Conference on Automated Deduction, Springer, pp. 748–752, doi:10.1007/3-540-55602-8_217.

[22] Lawrence C Paulson (1985): *Verifying the unification algorithm in LCF*. Science of computer programming 5, pp. 143–169, doi:10.1016/0167-6423(85)90009-7.

[23] John Alan Robinson (1965): *A Machine-Oriented Logic Based on the Resolution Principle*. Journal of the ACM 12(1), pp. 23–41, doi:10.1145/321250.321253.

[24] P. Y. A. Ryan & S. A. Schneider (1998): *An Attack on a Recursive Authentication Protocol. A Cautionary Tale*. Inf. Process. Lett. 65(1), p. 7–10, doi:10.1016/S0020-0190(97)00180-4.

[25] Christian Sternagel & René Thiemann (2018): *First-Order Terms*. https://isa-afp.org/entries/First_Order_Terms.html, Formal proof development.

[26] Coq Development Team: *The Coq Proof Assistant*. Available at https://coq.inria.fr/. Accessed: 2024-06-29.

[27] Max Tuengerthal, Ralf Küsters & Mathieu Turuani (2006): *Implementing a unification algorithm for protocol analysis with XOR*. arXiv preprint cs/0610014, doi:10.48550/arXiv.cs/0610014.

[28] Christian Urban, Andrew M Pitts & Murdoch J Gabbay (2004): *Nominal unification*. Theoretical Computer Science 323(1-3), pp. 473–497, doi:10.1016/j.tcs.2004.06.016.

[29] Rose Bohrer Yichi Xu, Daniel J Dougherty: *The Coq Proof Implementation*. Available at https://github.com/YiCXxxx/Coq-XORUnification. Accessed: 2024-06-29.

[30] Philipp Zech, Michael Felderer & Ruth Breu (2019): *Knowledge-based security testing of web applications by logic programming*. International Journal on Software Tools for Technology Transfer 21, pp. 221–246, doi:10.1007/s10009-017-0472-3.

Geospatial Trajectory Generation via Efficient Abduction: Deployment for Independent Testing

Divyagna Bavikadi Dyuman Aditya Devendra Parkar Paulo Shakarian
Arizona State University
Arizona, USA
dbavikad@asu.edu daditya1@asu.edu dparkar1@asu.edu pshak02@asu.edu

Graham Mueller
Leidos
Virginia, USA
William.G.Mueller@leidos.com

Chad Parvis
EpochGeo
Washington,D.C, USA
cp@epochgeo.com

Gerardo I. Simari
Universidad Nacional del Sur (UNS)
Bahía Blanca, Argentina
CONICET
Bahía Blanca, Argentina
gisimari@gmail.com

The ability to generate artificial human movement patterns while meeting location and time constraints is an important problem in the security community, particularly as it enables the study of the analog problem of detecting such patterns while maintaining privacy. We frame this problem as an instance of abduction guided by a novel parsimony function represented as an aggregate truth value over an annotated logic program. This approach has the added benefit of affording explainability to an analyst user. By showing that any subset of such a program can provide a lower bound on this parsimony requirement, we are able to abduce movement trajectories efficiently through an informed (i.e., A*) search. We describe how our implementation was enhanced with the application of multiple techniques in order to be scaled and integrated with a cloud-based software stack that included bottom-up rule learning, geolocated knowledge graph retrieval/management, and interfaces with government systems for independently conducted government-run tests for which we provide results. We also report on our own experiments showing that we not only provide exact results but also scale to very large scenarios and provide realistic agent trajectories that can go undetected by machine learning anomaly detectors.

1 Introduction

The ability to generate artificial human movement patterns while meeting location and time constraints is an important problem in the security community, particularly as it enables the study of the analog problem of detecting such patterns without the need for data from actual humans. This work is part of a larger effort to establish models of normal human movement at a fine-grain level[1] and operationalize those models and techniques in a system deployed to a government environment for evaluation. This contrasts with current techniques used to model populations that operate at a more coarse-grain level (country, county-level than building-level) as seen in previous work in specific areas such as population migration [4] or disease spread [7].

In this work, we focus on the generation of human movement patterns based on historical data. We frame this problem as an instance of abduction [23] in a geographic setting [29] guided by a novel parsimony function represented as an aggregate truth value over an annotated logic program [16]. This approach has the added benefit of affording explainability to an analyst user. By showing that any subset

[1]The IARPA HAYSTAC program, `https://www.iarpa.gov/research-programs/haystac`

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 274–287, doi:10.4204/EPTCS.416.24

of such a program can provide a lower bound on this parsimony requirement, we are able to abduce movement trajectories efficiently through an informed (i.e., A*) search. This fundamental result enables the practical implementation and deployment of software that is independently evaluated by a government test and evaluation (T&E) team. Specifically, in this paper, we make the following contributions.

1. We leverage the modularity of a logic program to guide informed search. Specifically, we provide a general result where we prove that a parsimony function consisting of an aggregate truth value of a logic program is bounded by such an aggregate of a subset of the program. In the case of our application, when rules correspond to one hop in an underlying graphical structure (in our case, a road network) we can provide an efficient graph-based heuristic based on above general result. We also provide empirical findings illustrating that the lower bound of the parsimony function for a subset logic program as well as show that an A* implementation provides improvements not available without a heuristic.
2. We build on the above strategy to provide scale through ad-hoc creation of the heuristic function that proceeds with the search. We provide empirical results demonstrating the scalability of the approach.
3. We show that our approach generates movement trajectories that are robust to machine learning (ML) models designed specifically to detect anomalous movements. We show the results of an internal test against such an ML model where we compare the output of the model with that of the training data and find in the vast majority of cases trajectories produced by our approach have anomaly ratings at or below the training data. We also illustrate how our approach allows for explainability of the anomalous portions of a trajectory by leveraging the deduction results of the logic program.
4. We describe how our approach can be integrated with rule mining, graph databases, and Amazon Web Services (AWS) cloud infrastructure in a system deployed for government testing. Further, we provide results of the government test where our approach was evaluated in four different environments against nine different machine learning models, all designed to find our trajectories. In the majority of cases, the ML models find our trajectories with a probability of detection (PD) of less than 0.40, which is the standard established by the government.

The remainder of the paper is organized as follows. In Section 2 we describe the application domain, and Section 3 provides a review of requisite logical preliminaries previously introduced in [16, 27, 3]. This is followed by Section 4, where we describe framing abduction for this domain by bounding the parsimony function to warrant informed search. The deployment is shown in Section 5. We report experimental results in Section 6 and discuss the findings of our internal evaluations where we show (in practice) tractable yet exact computation of parsimonious explanations, that movement trajectories provided are not susceptible to detection by machine learning-based anomaly detectors, and further scalability to larger graphs. In Section 7 we describe an independent evaluation with the probability of detection, Section 8 presents related work, and we conclude in Section 9 by outlining future research.

2 Motivating Application

In the aftermath of an unexpected event, such as a natural disaster, war, or large-scale industrial accident, human movement patterns can change significantly. As a result, IARPA (Intelligence Advanced Research Projects Activity) has identified problems relating to the characterization and generation of normal human movement patterns as a key problem of study in the HAYSTAC program[2]. In this problem, a given

[2]https://www.iarpa.gov/newsroom/article/finding-a-needle-with-haystac

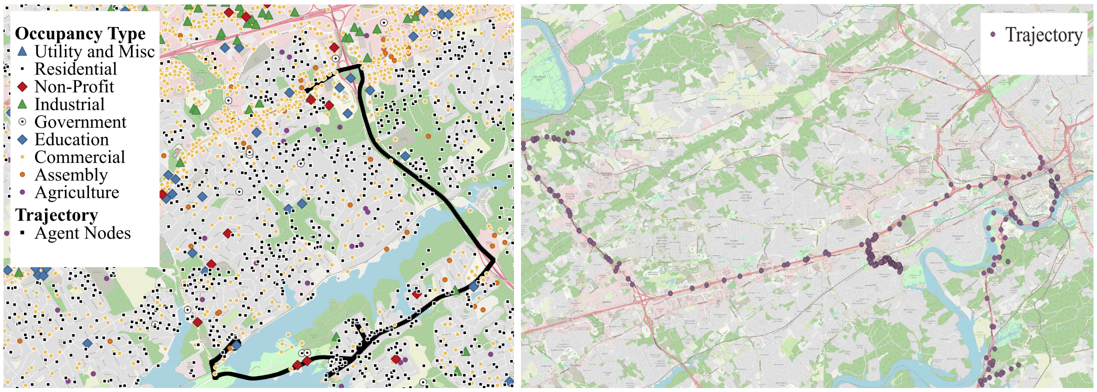


Figure 1: Left: The graph represents the city of Knoxville with landmarks, and the line plot denotes part of an agent's sub-trajectory. Right: Generated trajectory.

geospatial area is modeled as a graph where locations (we shall use $\mathscr{D}_{loc}$ to denote the domain of all possible locations) represent the vertices. A set of agents, e.g., $007, 008, .. \in \mathscr{D}_{agent}$, traverses through the network by using various modes of transportation, such as $personal\_vehicle \in \mathscr{D}_{movtype}$. Throughout the paper, we will use annotated logic [16] to specify various aspects of the environment by assigning predicates and then use temporal extensions [3]; temporal rules to specify the normalcy or abnormalcy of an agent's movement throughout the geospatial area. The rules will be learned from historical data, hence the abduction problem will consist of producing an agent trajectory (an explanation) between a start and end locations at certain times (observations) such that the abnormalcy of the trajectory is minimized (parsimony requirement). An example of a movement trajectory (before interpolation, discussed in Section 7) produced by our approach is shown in Figure 1(right) for the same agent whose training trajectory is shown in Figure 1(left). However, despite the specific application and ensuing deployment, we describe, our key insight is more general – if the parsimony function is specified as an aggregate over truth values resulting from the deductive process of a logic program, we can leverage the logic program to correctly improve efficiency (discussed in Section 4). This insight is what enables us to ultimately solve the problem at scale.

3 Technical Preliminaries

Syntax of Annotated Logic with Temporal Extensions. We use annotated logic syntax [16] with a lower-lattice based semantics [27] and temporal syntactic and semantic extensions [3]. 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 (i.e., $\mathscr{D}_i \subset \mathscr{C}$). Atoms are formed with terms (constants or variables) and predicates. Literals include atoms and their negation, with $\mathscr{G}$ being the set of all (ground) literals formed with no variables. Atoms and literals can be annotated with elements (intervals in $[0,1]$) of a lower semi-lattice structure $\mathscr{L}$ (not necessarily complete) with ordering $\sqsubseteq$. Here, to generalize fuzzy logic in the spirit of [22], we define the bottom element $\perp$ as $[0,1]$ and a set of top elements $\{[x,y] \mid x = y\}$. If $\mathbf{a}$ is an atom and μ is an annotation that denotes truth probability, then $\mathbf{a} : \mu$ is an *annotated atom*. We can create annotated negations (of atoms) in the same way, and together with annotated atoms this gives us the set of *annotated literals*. We also find it useful to extend the logic to make statements about time, and we do this in two different ways. First, following the convention of [28] we have *temporally annotated facts (TAFs)*. Given time point t and annotated literal f, f_t is a TAF that is true at t. The second extension is similar to that of

Allen's temporal logic [5] in which we have modal operators $\mathsf{AFTER}(f, f')$ and $\mathsf{BEFORE}(f, f')$, where f, f' are annotated literals, and the intuition is that f occurs after (resp., before) f'. Annotated literals and constructs formed with AFTER and BEFORE are called annotated formulas. Note that TAFs are considered separately from these formulas when we describe the semantics.

Example 3.1 (Language and Syntax) *We define specific domains as follows: $\mathscr{C}$ consists of disjoint sets $\mathscr{D}_{agent}, \mathscr{D}_{loc}, \mathscr{D}_{movtype}$. $\mathscr{D}_{agent}$ are constants associated with agents in the environment whose behavior we wish to model as mentioned in Section 2. We will have various predicates such as the binary predicate* `conn` *that takes elements of $\mathscr{D}_{loc}$ as arguments representing the connection of two locations. The truth values associated with ground atoms created with this predicate specify a road network (as depicted in Figure 1). We will also have a binary predicate,* `at` *specifying that an agent is at a certain location, e.g., for agent agent and location $loc \in \mathscr{D}_{loc}$, $\mathtt{at}(agent, loc)$ is a ground atom. We also have various unary predicates such as* `prim-Banks`*,* `occc-Education`*,* `occ-Residential`*,... (where prim encodes the primary use of the location like the building is primarily used as a Bank, and occ implies the occupancy type of the building like for Residential purpose) that take elements of $\mathscr{D}_{loc}$ as an argument. Finally, predicates of the form $\mathtt{anomalyType}(agent)$ are concluded from rules and are considered true with confidence indicated by its annotation if the agent is conducting abnormal behavior.*

Semantics and Satisfaction. Following previously introduced temporal extensions to annotated logic (i.e., [3, 25]), we assume a finite series of time points that we wish to reason about in an associated semantic structure of an interpretation I that, given timepoints $T = t_1, ..., t_{max}$, is any mapping $\mathscr{G} \times T \to \mathscr{L}$. The set $\mathscr{I}$ of all interpretations can be partially ordered via the ordering: $I_1 \preceq I_2$ iff for all ground literals $g \in \mathscr{G}$ and time t, $I_1(g,t) \sqsubseteq I_2(g,t)$. $\mathscr{I}$ forms a complete lattice under the $\preceq$ ordering. From this, we define a satisfaction relationship "$\models$" for temporally annotated facts in the usual manner (i.e., [16, 27, 3]).

Rules and Programs. We adopt the temporally extended *GAP rules* from [3]. If f_0 is an annotated atom and, $f_1, \ldots, f_m$ are annotated formulas , then

$$r \equiv f_0 \underset{\Delta t}{\longleftarrow} f_1 \wedge \ldots \wedge f_m \qquad \Delta t \geq 0$$

is called a *GAP rule*. When a conjunction of annotated formulas in the body is satisfied at time t, the annotation of the atom f_0 in the head gets updated after a delay, Δt. A GAP rule is called a *fact* when the body is empty, and *ground* when it has no occurrences of variables from $\mathscr{V}$. Table 1 shows examples of rules capturing anomalies that are learned based on observing an agent's routine for two weeks.

We define a *program*, Π, as a set of rules and TAFs. An interpretation I satisfies Π if for all $e \in \Pi, I \models e$. We also leverage a fixpoint operator presented in [27, 3] designed for use on lower-lattice annotated logic (which provides analogous results shown earlier for the operator introduced in [16]). The key here is that the operator provides a minimal model, hence exact answers to entailment under the assumption of consistency. As we learn logical rules from data, we can control consistency, so this is a reasonable assumption. We also use a fixpoint operator as per [16, 27, 3] to perform deductive inference. As per the previous work, fixpoint operator Γ is a map from interpretations to interpretations and is applied multiple times until convergence, denoted by Γ^*.

Abductive Inference. In this paper, we formalize an abduction problem as: given observations, represented as a set of TAFs denoted O, a set of hypotheses, which is also a set of TAFs denoted H, program Π, and parsimony function *value* that maps interpretations to positive reals, the goal is to identify an

Table 1: Rules with intuition for modeling the behavior of humans in a geospatial location

Rule	Natural Language
$abnormal(A):[0.9,1] \leftarrow_{\Delta t=0} education(A):[1,1] \wedge utility(A):[1,1] \wedge \mathsf{AFTER}(utility(A), education(A)):[1,1]$	If the agent A goes to a *utility* area like barns and sheds after visiting an *education* location (annotated by [1,1]) like a school, then it shows high abnormal activity by updating its lower bound to 0.9
$abnormal(A):[1,1] \leftarrow_{\Delta t=0} industrial(A):[1,1] \wedge assembly(A):[1,1] \wedge \mathsf{AFTER}(assembly(A), industrial(A)):[1,1]$	It is anomalous that an agent who works in a highly hazardous *industrial* location will directly go to a highly populated location like theaters (*assembly*)

explanation E, a subset of H such that $\Pi \cup E \cup O$ is consistent (in other words, $\Pi \cup E$ is consistent and entails O). Note that as both E and O are TAFs, we are not guaranteed consistency. Parsimony is measured by $value(\Gamma^{*}_{\Pi \cup E \cup O}(I_{\perp}))$ (where $I_{\perp}$ assigns all atoms at all time points to the annotation $[0,1]$, total uncertainty). An explanation with the lowest value for $value(\Gamma^{*}_{\Pi \cup E \cup O}(I_{\perp}))$ is an *optimal explanation*.

Example 3.2 *Building on the notion from example 3.1: in our use case, a set of observations O is a set of TAFs (essentially sets constraints on the agent's location at certain times), where each TAF indicates that an agent a is at $loc \in \mathscr{D}_{loc}$ at time t. H is the set of all possible locations $\mathscr{D}_{loc}$ of the agent. A set of learned temporally extended* GAP rules Π *indicate the normalcy of an agent in a graph G. Here, G is a series of TAFs that is formed with nodes from $\mathscr{D}_{loc}$. If $\mathtt{at}(a, loc) : \mu_t$ is a ground atom, then we impose graphical constraints like $\neg\mathtt{at}(a, loc') : \mu_{t+1}$, where $\mathtt{conn}(loc, loc') : \mu_t$ is not true in G and $loc \neq loc'$. As seen in Table 1 we use the lower bound of the annotation in the rule's head to represent the confidence of the body being historically abnormal. When the agent's movements relate to the body, its annotations are updated to* $[1,1]$ *for time t. Explanation E is a movement sequence (which is a set of TAFs) from loc^{i}_{start} to loc^{i}_{end} that fires most rules in* Π.

Framing the Abduction Problem as a Search Problem. While the general case of such an abduction problem is intractable, we have two key insights that apply to our domain problem. The first is a structural concern: if we are reasoning about an agent moving in a geospatial setting using a ground vehicle, we know that the agent cannot possibly teleport, so it is restricted to traveling along the graph at a certain speed. Similar restrictions have been applied to other problems such as social media diffusion [3], power grid failure modeling [18], and knowledge graph completion [21]. This allows for TAFs from set H to be selected in a sequential manner while at the same time limiting the TAFs to those consistent with the graphical structure, and the application of search routines such as depth-first search (which we implement in our experiments). However, we note that this does not reduce the branching factor enough to afford tractability. In the next section, we present results that allow for a provable lower bound on *value* in the general case, which we employ in our use case to obtain tractability and further scalability.

4 Efficient and Scalable Geospatial Abduction for Trajectory Generation

The complexity of logic-based abduction [9, 17] can be reduced by using a logic-based parsimony function instead of a standard logic-based function that is intractable due to the number of explanations. We introduce a parsimony function based on the aggregate truth values assigned using a logic program learned from data. The *value* obtains the lower bound of aggregate over the annotations on an atom b at time t for the minimal model I of $\Pi \cup E \cup O$. Using a lower bound on such an aggregate, we obtain an admissible heuristic allowing us to use informed search (i.e., A*). By extension, this addresses both intractability and scalability issues.

Bounding the Parsimony Function to enable Informed Search. We now provide new results that allow us to create a lower bound on the parsimony function *value* by taking a subset of the logic program. These rigorous results imply that we obtain an admissible and consistent heuristic function for A* – hence, the resulting usage of the lower bound of *value* in A* can provide an exact solution. Note that these are general results, not specific to the use case we are studying. However, clearly, their applicability depends on the subset of Π being non-trivial (e.g., the use of $\emptyset$ would be unhelpful). Further, the idea is that the subset of the logic program also offers a computational advantage. By showing for a given ground atom b, time t, and $\Pi' \subseteq \Pi$ and I, $\Gamma^*_{\Pi'}(I)(b,t) \sqsubseteq \Gamma^*_{\Pi}(I)(b,t)$, we state the following, which in turn gives us a lower bound on *value*:

Theorem 4.1 *For ground atom b, timepoint t, $\Pi' \subseteq \Pi$, and $I' \preceq I$, we have $\Gamma^*_{\Pi'}(I')(b,t) \sqsubseteq \Gamma^*_{\Pi}(I)(b,t)$.*

Informed Search Strategy. For our use case, a logic program Π is learned where the head of the rules is $\texttt{anomalyType}(agent)$ from Example 3.1. The body of the rule is determined by two major symbolic landmarks in G that are $n \in \mathbf{Z}^+$ hops away. Consider a single movement as moving and staying in a location (in our case, that is one hop away) from the current location during time t. For single hops, a subset of the logic program $\Pi^{SH} \subseteq \Pi$ is learned. In the general case, Theorem 4.1 shows that for a subset of the logic program $\Pi' \subseteq \Pi$, we get a lower bound on *value*. For a given set of movements I in G, we employ the *value* for a Π^{SH} as the heuristic function in an informed search strategy. We note that the increase in *value* resulting from single hop rules is inherently modular, meaning that for any node in the frontier set, such quantity is invariant. This allows us to precompute this increase for all nodes and store it in a graph-based data structure G_w. Considering the number of iterations of Γ as well as grounding for single movements versus a sequence of multiple movements, $\Gamma^*_{\Pi^{SH}}(I)$ is easier to compute, towards obtaining the heuristic value.

Scalable Heuristic Computation. Computation of *value* can be expensive on a whole logic program Π as it involves computing $\Gamma^*_{\Pi}(I)$ given a set of movements I as multiple anomalies can be inferred from multiple sequences of movements. The process of grounding can be expensive computationally, but we can gain efficiency by considering $\Pi^{SH} \subseteq \Pi$, where anomalous behavior rises from single movements. Using Theorem 4.1 we can efficiently prune abnormal candidate movements with informed search. We precompute the heuristic function by weighting the graph G with the lower bound on *value* for all possible single movements and obtain G_w. For further scalability, we compute *value* in a need-based fashion called *ad-hoc weighting* instead of precomputing it for all possible single movements in G. During heuristic computation we obtain the lower bound of *value* only when a certain movement is needed. For ad-hoc weighting, we compute *value* considering the agent's frontier up to a depth of 1 at each step.

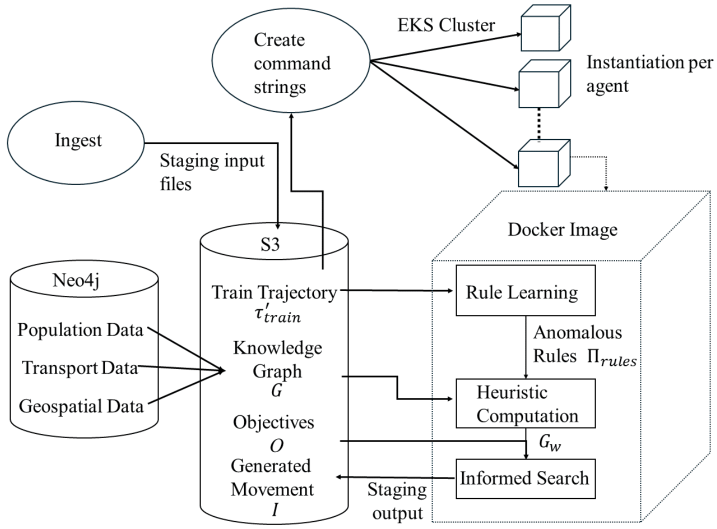


Figure 2: Visual representation of deployment.

5 Software Stack

Our logic program-guided abduction strategy is part of a software stack that was deployed in a cloud environment on Amazon Web Services (AWS) as per mandate by the government customer. Figure 2 depicts the overall system architecture (DAG structure).

Overall Workflow. The pipeline interfaces with the government system to access the raw geospatial data with related knowledge for 4 geolocations as well as training agent trajectories (as seen in Figure 1 (left)) and required objectives for each agent (framed as a set of observations O). The required objectives include typical human activities of single (visiting a friend, restaurant, etc) and recurring (going to work, etc.) movement types. Initially, we host the raw geospatial data in a Neo4j server and form a consolidated knowledge graph G (symbolic landmarks extracted from G are seen by the symbols in Figure 1 (left)). This is stored in an S3 bucket also containing the training data (consisting of both trajectories and objectives for each agent). For each agent, we extract spatial and temporal constraints from the training data. Based on a single trajectory of an agent, we learn a set of anomalous rules Π_{rules} (as seen in Table 1). Both G and Π_{rules} are used to compute the heuristic values by creating a weighted graph G_w. We also use constraints to perform an informed search algorithm to generate a normal trajectory (as seen in Figure 1 (right)).

Data Ingest. Our initial ingest and staging containerized processes are held in the DAG as nodes. Our ingest mechanism first parses the objective files for the agent to determine the locations of corresponding training trajectories. Secondly, based on the geolocation, we retrieve the appropriate knowledge graph and link it to each agent. Finally, we process the data associated with each agent (objectives, training data, graph) to a predefined staging area in the S3 bucket.

Instantiation. This step analyzes the staging folders and creates the necessary string commands specific to each agent. We launch pods for all agents with a movement-generation Docker image to process its particular objectives. As the container runs, generated movement instruction files are pushed to the appropriate output directory.

Rule Learning. To generate normal movement, we learn rules from the agent's historical data capturing realistic behavior. Sequences of movements deviating from this behavior are considered anomalous. Inferring from longer sequences can make the algorithm computationally expensive, but when we only compute a subset of rules involving shorter sequences, we can efficiently prune candidate movements that are highly abnormal using search algorithms. There are different kinds of rule types, which we call single-hop and multi-hop rules (Π_{SH}, Π_{MH}). Here, Π_{SH} leverages single movement frequency (for instance Table 1) while Π_{MH} leverages a set of multiple movements (in a similar format of Table 1). The logic program Π includes both types of rules. From Theorem 4.1, we can get $value(\Gamma^*_{\Pi^{SH} \cup E \cup O}(I)(b,t)) \sqsubseteq value(\Gamma^*_{\Pi \cup E \cup O}(I)(b,t))$ as $\Pi^{SH} \subseteq \Pi$; this is demonstrated in Figure 3.

Search. We use PyReason [3] to compute the fixpoint operator Γ used for both the actual calculation of value and the creation of the heuristic function. We use the aggregate function as an intersection over all annotations of the rules fired by Γ^*. This is computed in an ad-hoc fashion for possible movements, and we weight the graph based on those movements to form G_w. Given the set of constraints O, we form sub-abduction problems to perform an A* search, and generate normal trajectories satisfying all required objectives.

6 Internal Evaluation

Experimental Setup. The government provided us with simulated data we used for the internal development of our approach. We leverage three main kinds of data. Firstly, curated data for four geospatial locations: Knoxville, Singapore, Los Angeles, and San Francisco is collected from multiple source datasets, namely USAStructures [2], Planetsense [30] (which provides geospatial information), Open street map road network (which gives transportation information), Urbanpop [31] (containing population data), and other data collected as part of the program. Secondly, we have simulated trajectory data of 40,000 human agents across all 4 locations, which mimic realistic human activity. Four different teams each provide a different simulation environment for generating realistic training data. Note that this data comprises only location data and does not include information on actual people. Thirdly, we have a set of objectives for each agent that specifies spatio-temporal constraints. From the curated data, we use three types of input data: geospatial (building types, occupancy, etc), population (census data, population survey data, etc), and transportation (road network, statistics on traffic flow, etc) to build a consolidated knowledge graph G. Each node in G is either an intersection point from the road network or has attributes that convey its landmark category resembling building occupancy such as Commercial (stores, parking), Unclassified (does not require much security, non-residential), Non-Profit (general offices, Emergency Operation Centers), Residential (apartments, hotels), Assembly (convention centers, stadium), Education (libraries, schools), Utility (barns, water treatment), Industrial (hazardous factories- chemical factories, metal processing factories, construction), Agriculture (agricultural use land), Government (military, fire station). Each trajectory is a sequentially ordered tuple of length 2 weeks, consisting of latitude, longitude, and timestamp indicating the agent's location at each time point (cf. Figure 1 for an example of the trajectory on the graph). Moreover, every set of objectives describes spatial and temporal constraints on the trajectory to be generated. We generated 38 trajectories that satisfy all constraints for which the objectives were provided.

We conducted all experiments on a high-memory CPU machine with 128 cores, and 2000GB memory, using PyReason software [3] for inference. Rules (similar to Table 1) were learned using a bottom-up technique comparable to related work [26] where you restrict the body of the rule to contain historically possible single movements.

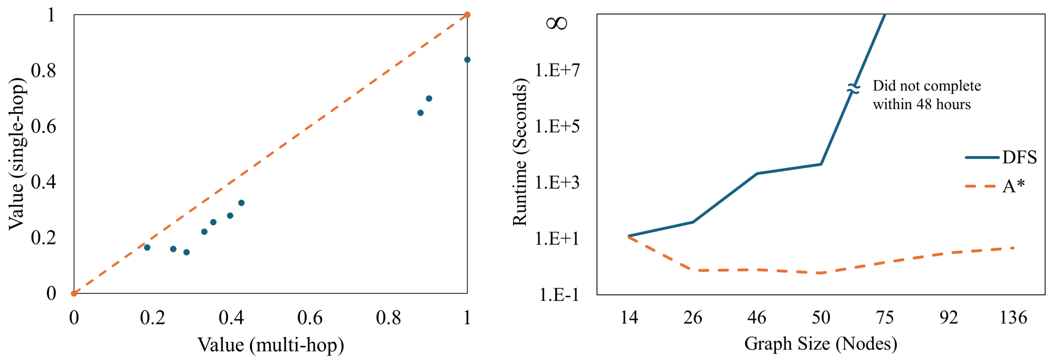


Figure 3: Left: Value with subset logic program, logic program Π', Π for the Knoxville location. Right: Runtime Comparison with Depth-First Search and A* Search.

Empirical Validation of Theoretical Claims. We present the results of two experiments as seen in Figure 3 to validate our claims concerning the use of the heuristic function and its employment as part of an informed search strategy. Figure 3 (left) shows that the heuristic value computed by the subset of the logic program is lower than the actual value in all our experiments since the data points lie below the dashed line – this aligns with Theorem 4.1. On the right panel, we see that the heuristic effectively addresses the tractability issues of the search, where the search conducted with the depth-first search algorithm without the heuristic could not be completed in under 48 hours even on small graphs with 50 nodes. Our approach using a heuristic maintains a lower running time as shown in Figure 3 (right).

Trajectory Robustness. A key aspect we wish to achieve with the generation of realistic movement trajectories is their robustness to anomaly detection. In this experiment, we run an ensemble of machine learning methods to perform anomaly detection over the generated and training trajectories to get their anomaly scores. The anomaly score of the ensemble is then compared with the average of that score over the historical trajectories of the agent, resulting in the relative anomaly ratio. In Figure 4 (left), we can see a box plot of anomaly scores of generated trajectories relative to the training data. In 90% of the data, a ratio lower than 1 is observed, indicating that mostly, anomalous movements identified in generated trajectories are no more frequent than those occurring in the training trajectories. For the cases where the anomaly detector found more anomalies than in the training data, the proportions are $46\%, 31\%, 14\%$, and 11%, which is likely to be lower than a practically employed high-precision threshold.

Scalability. We implemented both ad-hoc and non ad-hoc weighting for all 4 locations – see results in Figure 4 (right). Ad-hoc technique gave a maximum speedup of 245.42 and a minimum of 0.32 when compared to the non ad-hoc technique. Over all, ad-hoc weighting is beneficial due to reduced runtimes. Several outliers (not depicted in the plot) for 3 AOIs were observed with speedups of $128.05, 245.04$, and 140.46. Furthermore, when the graph size (the number of edges) increased by a factor of 4.69, the speedup was boosted by a factor of 72.05. The median speedups as the graph gets denser are $0.44, 1.30, 2.86$, and 17.98. These improvements show that the ad-hoc technique will scale well as the graph size grows significantly or gets denser when compared to non ad-hoc technique. In many cases, the non ad-hoc technique did not complete after running for more than 10 days, suggesting a speedup of over $1,000$x for those cases. Though such results are favorable to our approach, we did not consider these samples in the determination of the numbers in this section.

Explainability. Movemments in a trajectory are generated using a fixpoint-based algorithm [16] on Π.

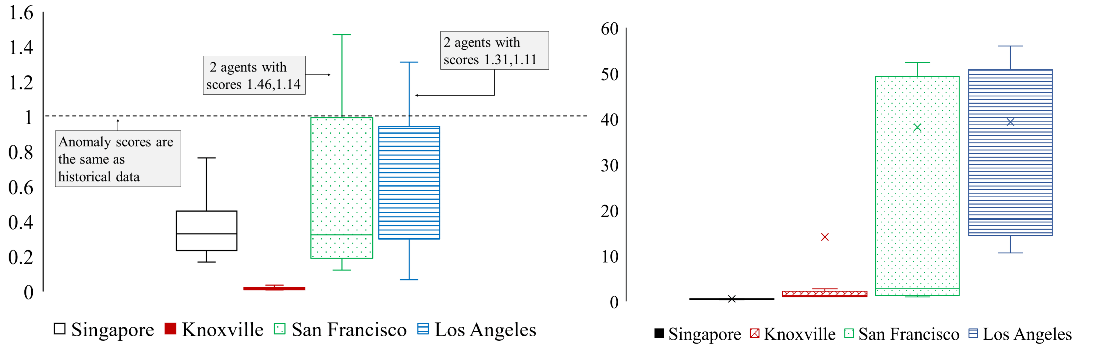


Figure 4: Left: Relative Anomaly Ratio. Anomalies in generated movements when compared with anomalies found in the training data by an ensemble ML anomaly detection algorithm. Right: Runtime speedup due to ad-hoc weighting.

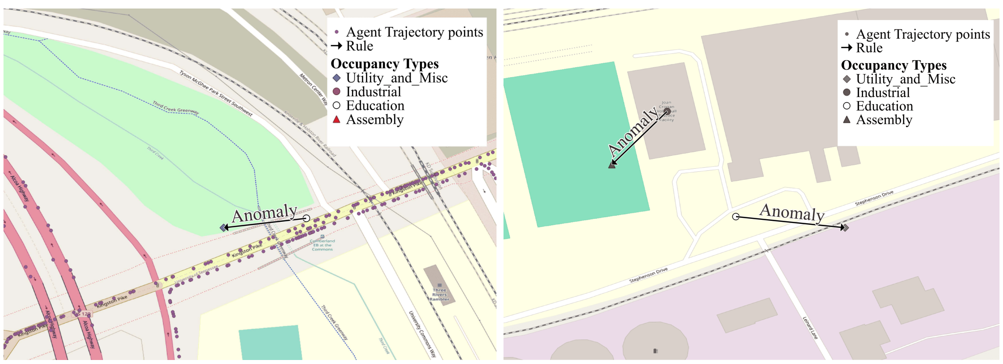


Figure 5: Visual depiction of rules from Table 1. Left: The top rule. Right: Both rules.

Movements tagged anomalous by a detected in such a trajectory can be backtracked to specific rules. Due to the modularity inherent to the approach, these rules assist analysts in assessing particular anomalous behaviors. An analyst can explore the anomalies detected on the generated trajectory by tracing back to the rules that caused such movements; for example, cf. Figure 5. Note that trajectory generation on historical data with our approach does not let an external entity infer an agent's identity. Even though the program is learned from the data, trajectory points in the agent's generated trajectory needn't be the same as historical data, which also contributes to privacy protection.

7 Deployment for Independent Evaluation

Independent Evaluation Setup. The generated trajectories are evaluated against nine different anomaly detection algorithms, each created by a government contractor. Further, these evaluations take place in the 4 locations examined in Section 6 but with newly simulated historical data for a duration of 2 months not previously available to our team. The results were collected and assessed by the government test and evaluation team using government-created cloud infrastructure to receive our outputs. The evaluation teams included an external T&E team, four simulation teams that run simulation engines to provide training data replicating real-world dynamics innate in human movement, and other teams that designed bespoke anomaly detection algorithms to evaluate the generated trajectories. Additionally, with the same

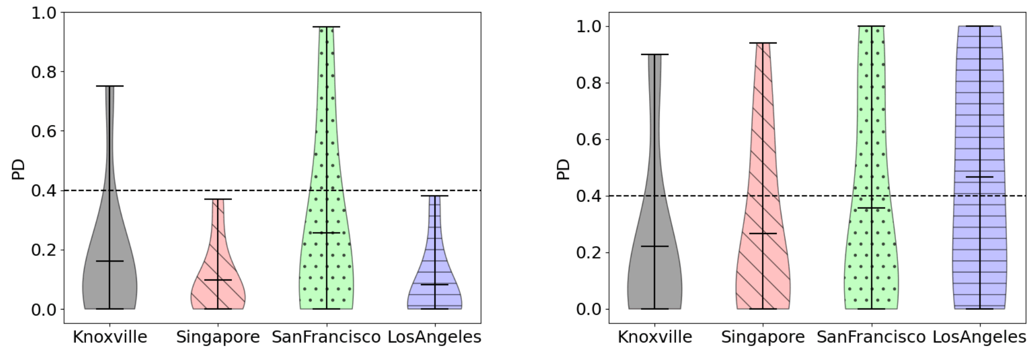


Figure 6: Prob. of detection (PD) from independent trials against ML-based anomaly detectors. Left: PD based on agents discovered. Right: PD based on points discovered.

simulators more movements are synthesized (which we are blind to) to be inserted with the 113 generated trajectories on the newly simulated historical data.

During this trial, we were evaluated on the probability of detection, defined as per the government's guidance [1]: $PD = \text{Correctly Detected Anomalies}/\text{All Anomalies}$. In Figure 6 we show the PD of generated movement trajectories against the various machine learning anomaly detectors developed in the program and for each of the four simulation environments. We note that several anomaly detectors were developed by teams that created simulation environments, which appeared to have an advantage We present results for F1-optimized anomaly detectors in Figure 6. It shows results curated by the government T&E team where anomalies are defined as anomalous agents being identified as well as when defined as points in the agent's trajectory. Note that our approach was designed to identify major intersections or landmarks that the agent would traverse. An interpolation procedure - not part of the overall route planning was used to fill in waypoints between such major intersections. This is likely why many anomaly detectors were able to identify anomalous points (Figure 6 right) but generally failed to find the agents from the overall trajectory (Figure 6 left). Improvement to the interpolation procedure is an area of future work. In the majority of cases, we met the government standard of having a PD below 0.40.

8 Related Work

Relation to Abduction Literature. Traditionally, in an abductive inference problem, an *explanation* is determined based on a model, which imposes some form of constraints, and a set of observations that though observed are phenomena that would be inferred from the model and explanation. The hardness of logic-based abduction (e.g., when the model is some form of logic program) is well established [9, 17], and a key source of this complexity is that the number of explanations is intractable in the general case. As a result, *parsimony* requirements are used to specify criteria over which explanation is preferable [23] – for example, previous work on geospatial abduction (but not abducing trajectories) minimized cardinality as a parsimony requirement [29]. In another example, recent applications of abduction to machine learning, such parsimony requirements are expressed in terms of training loss [12, 8]. Our work differs from all of the aforementioned papers in that we introduce a parsimony requirement based on the aggregate truth values assigned by a logic program that can be learned from data.

Relation to Literature on Geospatial Trajectory Generation. The abundance of trajectory data permits the analysis of mobility patterns and trajectories such as Markov-based paradigms have been applied for momentary goals (location prediction) [10, 19] and coarse-grain human movement analysis [32, 7, 4] - both different problems than this paper. Recent studies used recurrent neural networks (RNN) [15], generative adversarial networks (GAN) [11, 33], graph neural networks (GNN) [20], and transformer-based approaches [32] on real datasets to generate trajectories. Though these approaches tend to capture spatial-temporal correlations, they lack explainability. Recurrent-based approaches generate trajectories with short duration and spatial coverage. Adversarial approaches need more training data to get realistic outcomes while GNN-based models are time-consuming. A reinforcement-learning approach for sequence modeling [14, 6] can be extended for human trajectory generation but has challenges for large graphs. They heavily rely on the structure of the reward function. Interpretability has been induced using the latent space to see the distribution of uncertainty of semantic concepts [13]. Our approach can generate long-range trajectories spanning over a city with a length of two months, in a data-efficient manner [3] and scales to denser graphs. The modularity of our approach allows for detailed explanations.

9 Conclusion

In this paper, we described a system that generates realistic but synthetic human movement trajectories using abduction guided by a logic program. This system was recently deployed for independent government testing. We note that in our current iteration, there were several limitations. For example, we did not employ ad-hoc graph weighting (essentially a sequential operation) and parallelization together – finding how to achieve the right balance between the two techniques can result in further scalability. We also look to extend our logical language to provide additional insights into anomalies; for example, time of day is not currently considered, so an anomaly detector explicitly considering that aspect will readily find our trajectories. Further, we also look to leverage the ability of our underlying logic to accept arbitrary functional symbols to assign truth, allowing to leverage neurosymbolic techniques [24] to directly integrate ML anomaly detectors.

Ethics Statement. This work is part of the IARPA HAYSTAC program, which is designed to create simulated environments with generated human movement patterns not associated with actual persons and enable further study of human movement trajectories without relying on actual human data.

Acknowledgement This research is supported by the Intelligence Advanced Research Projects Activity (IARPA) via the Department of Interior/Interior Business Center (DOI/IBC) contract number 140D0423C0032. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DOI/IBC, or the U.S. Government. Also, this work was funded by ONR grant N00014-23-1-2580.

References

[1] *IARPA HAYSTAC*. `https://www.iarpa.gov/newsroom/article/haystac-baa-release`. Accessed: 05-08-2024.

[3] We had a single 2-week training sample per agent. For comparison, the ML approach of [11], uses on average 60 examples per agent that were 3 years in duration.

[2] *Oak Ridge National Laboratory (ORNL); Federal Emergency Management Agency (FEMA) Geospatial Response Office*. https://www.arcgis.com/home/item.html?id=0ec8512ad21e4bb987d7e848d14e7e24https://www.arcgis.com/home/item.html?id=0ec8512ad21e4bb987d7e848d14e7e24. Accessed: 05-08-2024.

[3] 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.

[4] C. Alis, E. F. Legara & C. Monterola (2021): *Generalized radiation model for human migration*. Sci Rep 11, p. 22707, doi:10.1038/s41598-021-02109-1.

[5] James F. Allen (1984): *Towards a general theory of action and time*. Artificial Intelligence 23(2), pp. 123–154, doi:10.1016/0004-3702(84)90008-0.

[6] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas & Igor Mordatch (2021): *Decision Transformer: Reinforcement Learning via Sequence Modeling*, doi:10.48550/arXiv.2106.01345. arXiv:2106.01345.

[7] Yuzhou Chen, Yulia R. Gel, Madhav V. Marathe & H. Vincent Poor (2024): *A simplicial epidemic model for COVID-19 spread analysis*. Proceedings of the National Academy of Sciences 121(1), p. e2313171120, doi:10.1073/pnas.2313171120. arXiv:https://www.pnas.org/doi/pdf/10.1073/pnas.2313171120.

[8] Wang-Zhou Dai, Qiuling Xu, Yang Yu & Zhi-Hua Zhou (2019): *Bridging machine learning and logical reasoning by abductive learning*. Advances in Neural Information Processing Systems 32. Available at https://proceedings.neurips.cc/paper_files/paper/2019/file/9c19a2aa1d84e04b0bd4bc888792bd1e-Paper.pdf.

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

[10] Sébastien Gambs, Marc-Olivier Killijian & Miguel Núñez del Prado Cortez (2012): *Next place prediction using mobility Markov chains*. In: Proceedings of the First Workshop on Measurement, Privacy, and Mobility, MPM '12, Association for Computing Machinery, New York, NY, USA, doi:10.1145/2181196.2181199.

[11] Lingyun Han (2024): *Enhanced Generation of Human Mobility Trajectory with Multiscale Model*. In Biao Luo, Long Cheng, Zheng-Guang Wu, Hongyi Li & Chaojie Li, editors: Neural Information Processing, Springer Nature Singapore, Singapore, pp. 309–323, doi:10.1007/978-981-99-8178-6_24.

[12] Yu-Xuan Huang, Wang-Zhou Dai, Yuan Jiang & Zhi-Hua Zhou (2023): *Enabling Knowledge Refinement upon New Concepts in Abductive Learning*. Proceedings of the AAAI Conference on Artificial Intelligence 37(7), pp. 7928–7935, doi:10.1609/aaai.v37i7.25959.

[13] Masha Itkina & Mykel Kochenderfer (2023): *Interpretable self-aware neural networks for robust trajectory prediction*. In: Conference on Robot Learning, PMLR, pp. 606–617, doi:10.48550/arXiv.2211.08701.

[14] Michael Janner, Qiyang Li & Sergey Levine (2021): *Offline Reinforcement Learning as One Big Sequence Modeling Problem*. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang & J. Wortman Vaughan, editors: Advances in Neural Information Processing Systems, 34, Curran Associates, Inc., pp. 1273–1286, doi:10.48550/arXiv.2106.02039. Available at https://proceedings.neurips.cc/paper_files/paper/2021/file/099fe6b0b444c23836c4a5d07346082b-Paper.pdf.

[15] Mahrokh Khakzar, Andry Rakotonirainy, Andy Bond & Sepehr G. Dehkordi (2020): *A Dual Learning Model for Vehicle Trajectory Prediction*. IEEE Access 8, pp. 21897–21908, doi:10.1109/ACCESS.2020.2968618.

[16] 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.

[17] Paolo Liberatore & Marco Schaerf (2015): *On the complexity of second-best abductive explanations*. International Journal of Approximate Reasoning 63, pp. 22–31, doi:10.1016/j.ijar.2015.05.009.

[18] X.-Q Liu, D.-Z Wang, C.-L Zhang & Y. Ning (2016): *Method of power grid fault diagnosis based on model prediction and abductive reasoning network* 37, pp. 472–476 and 480. doi:10.3969/j.issn.1005-3026.2016.04.004.

[19] X. Lu, E. Wetter, N. Bharti et al. (2013): *Approaching the Limit of Predictability in Human Mobility*. *Sci Rep 3* 2923, doi:10.1038/srep02923.

[20] Kai Lv & Liang Yuan (2023): *SKGACN: Social Knowledge-Guided Graph Attention Convolutional Network for Human Trajectory Prediction*. *IEEE Transactions on Instrumentation and Measurement* 72, pp. 1–11, doi:10.1109/TIM.2023.3283544.

[21] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli & Heiner Stuckenschmidt (2019): *Anytime Bottom-Up Rule Learning for Knowledge Graph Completion*. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, pp. 3137–3143, doi:10.24963/ijcai.2019/435.

[22] Nils J. Nilsson (1986): *Probabilistic logic*. *Artificial Intelligence* 28(1), pp. 71–87, doi:10.1016/0004-3702(86)90031-7. Available at `https://www.sciencedirect.com/science/article/pii/0004370286900317`.

[23] Yun Peng & James A. Reggia (1990): *Abductive inference models for diagnostic problem-solving*. Springer-Verlag New York, Inc., doi:10.1007/978-1-4419-8682-5.

[24] P. Shakarian, C. Baral, G.I. Simari & L. Pokala (2023): *Neuro Symbolic Reasoning and Learning*. SpringerBriefs in Computer Science, Springer Nature Switzerland, doi:10.1007/978-3-031-39179-8. Available at `https://books.google.com/books?id=hVEF0AEACAAJ`.

[25] Paulo Shakarian, Austin Parker, Gerardo Simari & Venkatramana Subrahmanian (2011): *Annotated probabilistic temporal logic*. *ACM Transactions on Computational Logic* 12(2), doi:10.1145/1877714.1877720.

[26] Paulo Shakarian, Austin Parker, Gerardo Simari & Venkatramana V. S. Subrahmanian (2011): *Annotated probabilistic temporal logic*. *ACM Trans. Comput. Logic* 12(2), doi:10.1145/1877714.1877720.

[27] Paulo Shakarian & Gerardo I. Simari (2022): *Extensions to Generalized Annotated Logic and an Equivalent Neural Architecture*. In: *Proceedings - 2022 4th International Conference on Transdisciplinary AI, TransAI 2022*, Institute of Electrical and Electronics Engineers Inc., pp. 63–70, doi:10.1109/TransAI54797.2022.00017.

[28] Paulo Shakarian, Gerardo I. Simari & V. S. Subrahmanian (2012): *Annotated Probabilistic Temporal Logic: Approximate Fixpoint Implementation*. *ACM Trans. Comput. Logic* 13(2), doi:10.1145/2159531.2159535.

[29] Paulo Shakarian, V. S. Subrahmanian & Maria Luisa Sapino (2011): *GAPs: Geospatial Abduction Problems*. *ACM Trans. Intell. Syst. Technol.* 3(1), doi:10.1145/2036264.2036271.

[30] Gautam Malviya Thakur, Budhu Bhaduri, Jesse Piburn, Kelly Sims, Robert Stewart & Marie Urban (2015): *PlanetSense: A Real-time Streaming and Spatio-temporal Analytics Platform for Gathering Geo-spatial Intelligence from Open Source Data* (1). doi:10.1145/2820783.2820882.

[31] Joseph Tuccillo, Robert Stewart, Amy Rose, Nathan Trombley, Jessica Moehl, Nicholas Nagle & Budhendra Bhaduri (2023): *UrbanPop: A spatial microsimulation framework for exploring demographic influences on human dynamics*. *Applied Geography* 151, p. 102844, doi:10.1016/j.apgeog.2022.102844.

[32] Yu Wang, Tongya Zheng, Yuxuan Liang, Shunyu Liu & Mingli Song (2024): *COLA: Cross-city Mobility Transformer for Human Trajectory Simulation*. In: *Proceedings of the ACM on Web Conference 2024, WWW '24* 41, ACM, p. 3509–3520, doi:10.1145/3589334.3645469.

[33] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong & Wancai Zhang (2021): *Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting*. *Proceedings of the AAAI Conference on Artificial Intelligence* 35(12), pp. 11106–11115, doi:10.1609/aaai.v35i12.17325. Available at `https://ojs.aaai.org/index.php/AAAI/article/view/17325`.

Towards Mass Spectrum Analysis with ASP (Extended Abstract)

Nils Küchenmeister
TU Dresden
Dresden, Germany
nils.kuechenmeister@tu-dresden.de

Alex Ivliev
TU Dresden
Dresden, Germany
alex.ivliev@tu-dresden.de

Markus Krötzsch
TU Dresden
Dresden, Germany
markus.kroetzsch@tu-dresden.de

We present a new use of Answer Set Programming (ASP) to discover the molecular structure of chemical samples based on mass spectrometry data. To constrain the exponential search space for this combinatorial problem, we develop canonical representations of molecular structures and an ASP implementation that uses these definitions. This implementation forms the core of our new tool, GENMOL, designed to enumerate potential molecular structures given a specified composition of fragments.

1 Introduction

Mass spectrometry is a powerful technique to determine the chemical composition of a substance. However, the mass spectrum of a substance does not reveal its exact molecular structure, but merely the possible ratios of elements in the compound and its fragments. To identify a sample, researcher may use commercial databases (for common compounds), or software tools that can discover molecular structures from the partial information available. The latter leads to a combinatorial search problem that is a natural fit for answer set programming (ASP). Molecules can be modeled as undirected graphs, representing the different elements and atomic bonds as node and edge labels, respectively. ASP is well-suited to encode chemical domain knowledge (e.g., possible number of bonds for carbon) and extra information about the sample (e.g., that it has an *OH* group), so that each answer set encodes a valid molecular graph.

Unfortunately, this does not work: a direct ASP encoding yields exponentially many answer sets for each molecular graph due to the large number of symmetries (automorphisms) in such graphs. For example, $C_6H_{12}O$ admits 211 distinct molecule structures but leads to 111,870 answer sets. Removing redundant solutions and limiting the search to unique representations are common techniques used in the ASP community where they have motivated research on *symmetry breaking*. Related approaches work by adding additional rules to ASP [1, 4, 9], by rewriting the ground program before solving [5, 3, 2], or by introducing dedicated solvers [7]. However, our experiments with some of these approaches still produced 10–10,000 times more answer sets than molecules even in simple cases.

In this extended abstract, we summarize our new approach that prevents symmetries in graph representations already during grounding [8]. This technique forms the basis of our ASP-driven prototype implementation for enumerating molecular structures using partial chemical information. Our ASP source code, evaluation helpers, and data sets are available online at `https://github.com/knowsys/eval-2024-asp-molecules`. The sources of our prototype application are at `https://gitlab.com/nkuechen/genmol/`.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 288–290, doi:10.4204/EPTCS.416.25

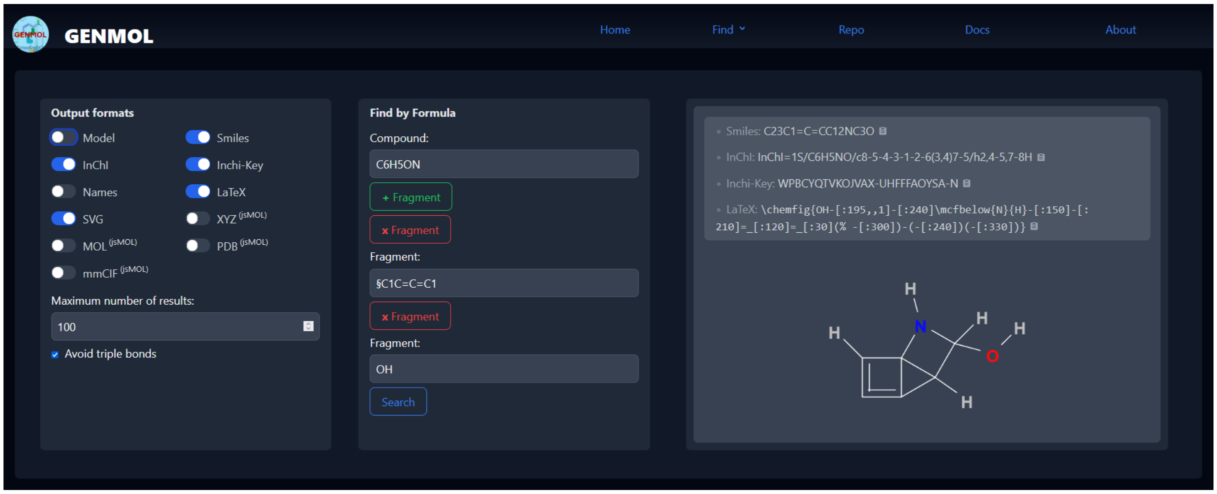


Figure 1: User interface of GENMOL

2 Analysis of Mass Spectra with Genmol

Many mass spectrometers break up samples into smaller fragments and measure their relative abundance. The resulting mass spectrum forms a characteristic pattern, enabling inferences about the underlying sample. High-resolution spectra may contain information such as "the molecule has six carbon atoms" or "there is an *OH* group", but cannot reveal the samples's full molecular structure. In chemical analysis, we are looking for molecular structures that are consistent with the measured mass spectrum. To address this task, we have developed GENMOL, a prototype application for enumerating molecular structures for a given composition of fragments. It is available as a command-line tool and as a progressive web application (PWA), shown in Fig. 1. GENMOL is implemented in Rust, with the web front-end using the Yew framework on top of a JSON-API, whereas the search for molecular structures is implemented in Answer Set Programming (ASP) and solved using *clingo* [6]. An online demo of Genmol is available for review at `https://tools.iccl.inf.tu-dresden.de/genmol/`.

The screenshot shows the use of GENMOL with a *sum formula* C_6H_5ON and two fragments as input. Specifying detected fragments and restricting bond types helps to reduce the search space. Alternatively, users can provide a molecular mass or a complete mass spectrum, which will then be associated with possible chemical formulas using, e.g., information about the abundance of isotopes. The core task of GENMOL then is to find molecules that match the given input constraints. Molecules in this context are viewed as undirected graphs of atoms, linked by covalent bonds that result from sharing electrons.[1] Many chemical elements admit a fixed number of bonds, the so-called *valence*, according to the number of electrons available for binding (e.g., carbon has a valence of 4). Bonds may involve several electrons, leading to single, double, triple bonds, etc. The graph structure of molecules, the assignment of elements, and the possible types of bonds can lead to a large number of possible molecules for a single chemical formula, and this combinatorial search task is a natural match for ASP.

[1] This graph does not always determine the spacial configuration of molecules, which cannot be determined by mass spectrometry alone, yet it suffices for many applications.

3 Canonical Representation of Molecules

Our experiments show that the existing approaches to symmetry breaking outlined in the introduction do not achieve adequate performance for real-world use cases [8]. Our dedicated encoding therefore relies on a canonical representation of molecules, which restricts the overall search space. It is based on SMILES, a widely used serialization format for molecular graphs. SMILES start from an (arbitrary) spanning tree of the molecular graph, serialized in a depth-first order, with subtrees enclosed in parentheses. Edges not covered by the spanning tree are indicated by pairs of numeric cycle markers.

The ASP implementation can then be summarized as follows: (1) guess the spanning tree, defined by the element of each node, the parent-relation between nodes, and the bond type of the parent edge of each node, (2) guess the cycle markers to complete the molecule, (3) eliminate non-canonical solutions.

Canonical Molecular Trees We first consider the acyclic case. We compare tree nodes by considering the (1) maximum depth and number of vertices of its subtree, (2) number of children, and (3) bond type of its parent edge. This ordering extends to trees by comparing their roots and recursively descending on ties. The minimal tree according to this ordering results in a unique representation.

Canonical Molecular Graphs We reduce the general case to the acyclic case by introducing edges to fresh nodes for each cycle marker. Since we cannot compare every such representation in practice, we implement a heuristic, which disallows *shortening cycle edges*. Intuitively, such cycles occur if another assignment of cycle markers would lead to a deeper tree. The ASP implementation detects shortening cycle edges by a pattern-matching approach.

References

[1] Michael Codish, Alice Miller, Patrick Prosser & Peter J. Stuckey (2019): *Constraints for symmetry breaking in graph representation*. *Constraints* 24(1), pp. 1–24, doi:10.1007/s10601-018-9294-5.

[2] Jo Devriendt & Bart Bogaerts (2016): *BreakID: Static Symmetry Breaking for ASP (System Description)*. *CoRR* abs/1608.08447. Available at `http://arxiv.org/abs/1608.08447`.

[3] Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe & Marc Denecker (2016): *Improved Static Symmetry Breaking for SAT*. In: *Proc. 19th Int. Conf. Theory and Applications of Satisfiability Testing (SAT'16)*, *LNCS* 9710, Springer, pp. 104–122, doi:10.1007/978-3-319-40970-2_8.

[4] Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe & Marc Denecker (2016): *On local domain symmetry for model expansion*. *Theory Pract. Log. Program.* 16(5-6), pp. 636–652, doi:10.1017/S1471068416000508.

[5] Christian Drescher, Oana Tifrea & Toby Walsh (2011): *Symmetry-breaking answer set solving*. *AI Communications* 24(2), pp. 177–194, doi:10.3233/AIC-2011-0495.

[6] Martin Gebser, Benjamin Kaufmann, André Neumann & Torsten Schaub (2007): *Conflict-Driven Answer Set Solving*. In: *Proc. 20th Int. Joint Conf. on AI (IJCAI'07)*, p. 386, doi:10.1016/j.artint.2012.04.001.

[7] Tarek Khaled & Belaid Benhamou (2018): *Symmetry breaking in a new stable model search method*. In: *LPAR-22 Workshop and Short Paper Proceedings*, *Kalpa Publications in Computing* 9, EasyChair, pp. 58–74, doi:10.29007/1l5r.

[8] Nils Küchenmeister, Alex Ivliev & Markus Krötzsch (2024): *Towards Mass Spectrum Analysis with ASP*. In: *Proc. Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'24)*, Springer. To appear.

[9] Alice Tarzariol, Martin Gebser, Konstantin Schekotihin & Mark Law (2023): *Learning to Break Symmetries for Efficient Optimization in Answer Set Programming*. In: *Proc. 35th AAAI Conf. on Artificial Intelligence (AAAI'23)*, AAAI Press, pp. 6541–6549, doi:10.1609/aaai.v37i5.25804.

Monitoring and Scheduling of Semiconductor Failure Analysis Labs

Elena Mastria
University of Calabria, Italy
elena.mastria@unical.it

Domenico Pagliaro
KAI GmbH, Austria
University of Klagenfurt, Austria
domenico.pagliaro@k-ai.at

Francesco Calimeri
University of Calabria, Italy
DLVSystem Srl, Italy
francesco.calimeri@unical.it

Simona Perri
University of Calabria, Italy
simona.perri@unical.it

Martin Pleschberger
KAI GmbH
martin.pleschberger@k-ai.at

Konstantin Schekotihin
University of Klagenfurt, Austria
konstantin.schekotihin@aau.at

Identifying and locating non-conformities, such as physical failures causing electrical malfunctioning of a device, in modern semiconductor devices is a challenging task. Typically, highly qualified employees in a failure analysis (FA) lab use sophisticated and expensive tools like scanning electron microscopes to identify and locate such non-conformities. Given the increasing complexity of investigated devices and very limited resources, labs may struggle to deliver analysis results in time. This paper suggests an approach to optimize the usage of FA lab resources by adaptive scheduling and monitoring. In particular, we combine constraints programming for the computation of a schedule with stream reasoning to monitor the lab's conditions and maintain the schedule depending on the situation. Evaluation results indicate that our system can significantly improve the tardiness of a real-world FA lab, and all its computational tasks can be finished in an average time of 3.6 seconds, with a maximum of 15.2 seconds, which is acceptable for the lab's workflows.

1 Background and Proposal

Failure Analysis (FA) of semiconductor devices is an important activity aiming at the identification of non-conformities during various stages of the device's lifecycle. For instance, an FA lab might provide qualifications for new designs, execute tests for quality management tasks, or identify failures in malfunctioning devices returned by customers. Daily, FA labs process a large number of jobs and, given their complexity, often face issues with delivering all analysis results on time. To minimize the likelihood of violating deadlines, a lab must optimally utilize all available resources, such as equipment and personnel. At the moment, First-In-First-Out (FIFO) in combination with job prioritization is the most widely used strategy for assigning FA resources. Although FIFO can easily be implemented in a lab, its long-term performance is unsatisfactory, since it handles poorly situations when jobs have different priorities or some of the resources have limited availability. To solve this issue, researchers suggested various methods for modeling and solving the scheduling problem for different types of labs, such as medical or industrial ones. In the literature, researchers addressed different aspects of laboratory scheduling reflecting the complex and dynamic nature of the environment. The papers present scheduling models usually derived from two classic problems: job-shop scheduling (JSS), e.g., [7, 5], or resource-constrained project scheduling (RCPS), e.g., [4, 3]. In addition to modeling aspects, designers of these approaches investigated efficient solving methods able to compute high-quality schedules within a given time budget, thus, avoiding delays while interacting with lab employees over graphical interfaces. For instance, [4] suggest engineers precisely plan their daily tasks and provide updates to the schedule in case of any

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 291–294, doi:10.4204/EPTCS.416.26

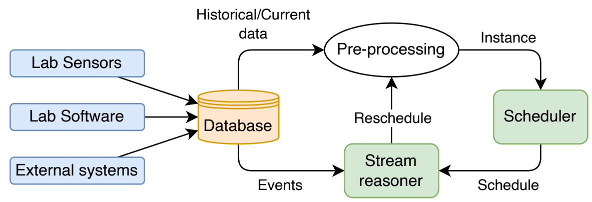


Figure 1: General architecture of the system.

unforeseen events. High-performance scheduling algorithms of the system ensure the responsiveness of a scheduling system during this interaction.

The main drawback of existing solutions is that they focus solely on the scheduling aspect of the problem and require users to monitor the lab's dynamic environment characterized by a large number of stochastic events, such as equipment breakdowns, personnel emergencies, or unexpected task delays. All these events must be timely and correctly communicated to the scheduling system to ensure its correct operation. However, manual maintenance of the work schedule by lab engineers is time-consuming and might be a source of erroneous data causing incorrect scheduling results. To overcome these issues, we propose a novel approach that aims to avoid the aforementioned issues by automatic monitoring, repair, and (re)scheduling of lab tasks. Our approach relies on two components: a *stream reasoner* based on *DP*-sr [2] and a *scheduler* using the constraints programming approach of [5].

DP-sr supports an input language that properly extends Answer Set Programming (ASP) with ad-hoc means for dealing with Stream Reasoning (SR) scenarios, yet maintaining the highly declarative nature and ease of use that are signature features of ASP. Along with typical ASP features (e.g., aggregates, disjunction, strong/weak constraints, unstratified negation), specific constructs can appear in *DP*-sr programs that are supposed to be evaluated over a *stream* of events. The scheduler is implemented using IBM ILOG Cplex Optimization Studio.[1] This constraints programming framework is widely used in the scheduling of production processes in the semiconductor industry. The framework provides a versatile knowledge representation language with specific constraints designed for efficient encoding of scheduling problems and a powerful solver able to efficiently find optimal solutions for a given instance.

Intuitively, in our solution the stream reasoner monitors the lab's events to ensure that the current situation corresponds to the existing schedule. Whenever violations are detected the reasoner tries to automatically repair the schedule or, otherwise, notifies the scheduler about the violation. Given the current situation in a lab, the scheduler computes an assignment of tasks to engineers and machines optimizing the selected measure, such as tardiness or makespan. The former is often preferred since it corresponds to the popular key performance indicator of an FA lab. We evaluated the system on simulated re-scheduling scenarios derived from logs of real-world FA labs. Obtained results show that our method averages 6.8 seconds for each reasoning cycle, ranging from 1 to 15 seconds, including monitoring (0.8-3.3 seconds) and scheduling stages (2.1-13.4 seconds). This is significantly faster than the standard tasks in FA Lab, which usually take over 20 minutes on average. [8].

2 Problem and Data

Scheduling refers to the process of assigning and managing the execution of tasks, given a set of resources. Following [5], we model the problem as a variant of the JSS problem with the following ex-

[1] `https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer`

tensions: *multi-resource* - more than one resource type, e.g., engineers or machines [6], and *flexible* - there are multiple instances of every resource. [1]. The resulting JSS variant is an NP-hard optimization problem, where the aim is to find an optimal allocation of resources to tasks and time points, such that: (i) each resource instance is assigned to only one task at a time; (ii) every task gets all required resources; (iii) execution of tasks is non-preemptive; (iv) no tasks of a job are executed in parallel; and (v) tasks must be scheduled w.r.t. their order. Two of the possible objective functions that can be implemented when evaluating the effectiveness of a scheduling system are *tardiness* that sums up all job delays, given their deadlines, and *makespan* that measures the total time required to complete the jobs. In this work, we encode the scheduling problem using constraints programming and employ tardiness as the optimization criterion since FA labs often use the fulfillment of deadlines as a key performance indicator. Nevertheless, we also report the makespan in the evaluation results.

Our main data source consists of an anonymized slice of the main data source of an FA database comprising 1646 jobs over 3 years, from January 2020 to December 2022.[2] The dataset provides all the information needed by the scheduler to compute the optimal task assignments and make recommendations in a simulated environment.

3 General Architecture

In the following, we provide an overview of the system; the general architecture is depicted in Figure 1. At first, the initial dataset is pre-processed, and fed to the scheduler for the computation of a new schedule. Along with the schedule, the system produces plots of the scheduled tasks and information about the overall tardiness. The monitoring component is set up as soon as the scheduled execution starts. More in detail, the stream reasoner *DP*-sr is fed with the *DP*-sr program, corresponding to the monitoring task to perform over the FA events, along with the background knowledge, representing the available resources, task-resources compatibility, and job deadlines. From now on, *DP*-sr will be waiting for schedule information and/or timestamped events coming from the database, and an interaction loop starts between the scheduler and the monitor managed by the Python application. At the first iteration, *DP*-sr is provided with the schedule previously computed along with possible events occurring at the beginning; subsequently, iterations are executed at a predefined frequency. At each iteration, all events that occurred after the previous iteration are fed to *DP*-sr, that might also be fed a new schedule, if a reschedule was needed. Each time *DP*-sr receives the data, it performs its computation across all timestamps featured by the events of the current iteration, providing an answer for all included timestamps. The *DP*-sr answer is analyzed and classified into one the following categories. *Propose of new resource(s) in case of conflicts.* When there is no need of a full reschedule, *DP*-sr suggests a resource replacement based on the conflicts that could happen if one or more tasks are delayed. The proposal of a new machine and/or a new worker is considered before changing the whole schedule. The new resources are inserted into the current schedule and an acknowledgment is sent back to *DP*-sr, such that the scheduled events are updated. Plots are as well generated highlighting the changes in the schedule. *Irreparable conflicts.* When conflicts are not solvable, *DP*-sr identifies it and notifies the need of a complete reschedule. Therefore, a reschedule takes place and a new schedule is computed and sent back to *DP*-sr, so to update the knowledge base. The system evaluates the new schedule defining a new tardiness and new plots are generated. *No conflicts found.* In this case, there are no errors or conflicts and the system keeps running with no exceptions.

[2]https://zenodo.org/records/10069426

References

[1] P. Brucker & R. Schlie (1990): *Job-shop scheduling with multi-purpose machines*. *Computing* 45(4), pp. 369–375, doi:10.1007/BF02238804. Available at `http://link.springer.com/10.1007/BF02238804`.

[2] Francesco Calimeri, Elena Mastria & Simona Perri (2024): *DP-SR: a purely Declarative Programming framework for Stream Reasoning*. Available at `https://sites.google.com/unical.it/dp-sr`.

[3] Mariana M. Cunha, Joaquim L. Viegas, Miguel S.E. Martins, Tiago Coito, Andrea Costigliola, João Figueiredo, João M.C. Sousa & Susana M. Vieira (2019): *Dual Resource Constrained Scheduling for Quality Control Laboratories*. *IFAC-PapersOnLine* 52(13), pp. 1421–1426, doi:10.1016/j.ifacol.2019.11.398. Available at `https://www.sciencedirect.com/science/article/pii/S2405896319313795`. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

[4] Philipp Danzinger, Tobias Geibinger, David Janneau, Florian Mischek, Nysret Musliu & Christian Poschalko (2023): *A System for Automated Industrial Test Laboratory Scheduling*. *ACM Trans. Intell. Syst. Technol.* 14(1), doi:10.1145/3546871.

[5] Giulia Francescutto, Konstantin Schekotihin & Mohammed M. S. El-Kholany (2021): *Solving a Multi-resource Partial-Ordering Flexible Variant of the Job-Shop Scheduling Problem with Hybrid ASP*. In Wolfgang Faber, Gerhard Friedrich, Martin Gebser & Michael Morak, editors: *Logics in Artificial Intelligence - 17th European Conference, JELIA 2021, Virtual Event, May 17-20, 2021, Proceedings*, *Lecture Notes in Computer Science* 12678, Springer, pp. 313–328, doi:10.1007/978-3-030-75775-5_21.

[6] V. B. Gargeya & R. H. Deane (1996): *Scheduling research in multiple resource constrained job shops: a review and critique*. *International Journal of Production Research* 34(8), pp. 2077–2097, doi:10.1080/00207549608905015.

[7] Takeshi D. Itoh, Takaaki Horinouchi, Hiroki Uchida, Koichi Takahashi & Haruka Ozaki (2021): *Optimal Scheduling for Laboratory Automation of Life Science Experiments with Time Constraints*. *SLAS Technology* 26(6), pp. 650–659, doi:10.1177/24726303211021790. Available at `https://www.sciencedirect.com/science/article/pii/S2472630322011359`. Special Collection: Assay Guidance Manual for Drug Discovery: Technologies that Matter.

[8] Domenico Pagliaro, Martin Pleschberger, Olivia Pfeiler, Thomas Freislich & Konstantin Schekotihin (2023): *Working Time Prediction and Workflow Mining at Failure Analysis*. In: *ISTFA*, Phoenix, Arizona, USA, pp. 121–130, doi:10.31399/asm.cp.istfa2023p0121. Available at `https://dl.asminternational.org/istfa/proceedings/ISTFA2023/84741/121/28574`.

Declarative AI design in Unity using Answer Set Programming (Extended Abstract)*

Denise Angilica† Giovambattista Ianni† Francesco Pacenza† Jessica Zangari†

Declarative methods such as Answer Set Programming show potential in cutting down development costs in commercial videogames and real-time applications in general. Many shortcomings, however, prevent their adoption, such as performance and integration gaps. In this work we illustrate *ThinkEngine*, a framework in which a tight integration of declarative methods within the typical game development workflow is made possible in the context of the Unity game engine.

Introduction. The AI research field shares a past and a present of reciprocal exchange with the game design industry, both whether we are talking of inductive/machine learning-based techniques or knowledge-based, deductive techniques. In this context, machine learning is useful in several respects but still presents some limitations like the fact that it is not easily "tunable" and configurable at will, and has non-negligible design-time costs. In order to overcome those limits, one can consider the introduction of declarative knowledge representation techniques, which offer better explainability potential and, especially, are *elaboration tolerant* in the sense of McCarthy [4]. However, known performance and integration shortcomings limited so far the usage of these techniques in videogames. Among declarative approaches, Answer Set Programming (ASP) has been experimentally used in videogames to various extents, such as for declaratively generating level maps [6] and for the definition of artificial players [2] for the Angry Birds AI competition. ASP makes no exception with respect to the shortcomings that restrict the utility of declarative approaches. Two obstacles are of concern: performance in real-time contexts and integration, i.e., ease of wiring with other standard parts of the game logic.

ThinkEngine is an asset working for the popular Unity game engine, aiming at narrowing the above gaps. Our tool enables the possibility of introducing ASP-based reasoning modules within Unity-made games. The key contributions of *ThinkEngine* are the following:

-*It integrates declarative AI modules in game engines*. This requires to overcome several obstacles, as one has to coordinate the interaction of AI modules in the so called iterative "game loop". It must be noted that the game loop, the typical execution paradigm of a videogame run-time, is an exemplary real-life usage of the sense-think-act cycle commonly deployed in agents and robotic systems [5]. *ThinkEngine* introduces a hybrid/deliberative scheme, transparently using auxiliaries threads in order to offload time consuming reasoning tasks.

-*AI modules can be either reactive or deliberative*. Multiple competing plans can be generated outside of the main game loop; plans can be selected according to programmable priorities, and can be aborted, replaced or restarted on a per action basis.

-*An incremental or a general solver of choice can be used for evaluating logic programs*. Given it is expected that reasoning tasks are iteratively repeated in videogames, it is possible to use [3], or a general ASP solver.

*This work was partially supported by the PNRR MUR project PE0000013-FAIR, Spoke 9 - Green-aware AI – WP9.1 and by the LAIA laboratory (part of the SILA laboratory network at University of Calabria).

†University of Calabria, name.surname@unical.it

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 295–297, doi:10.4204/EPTCS.416.27

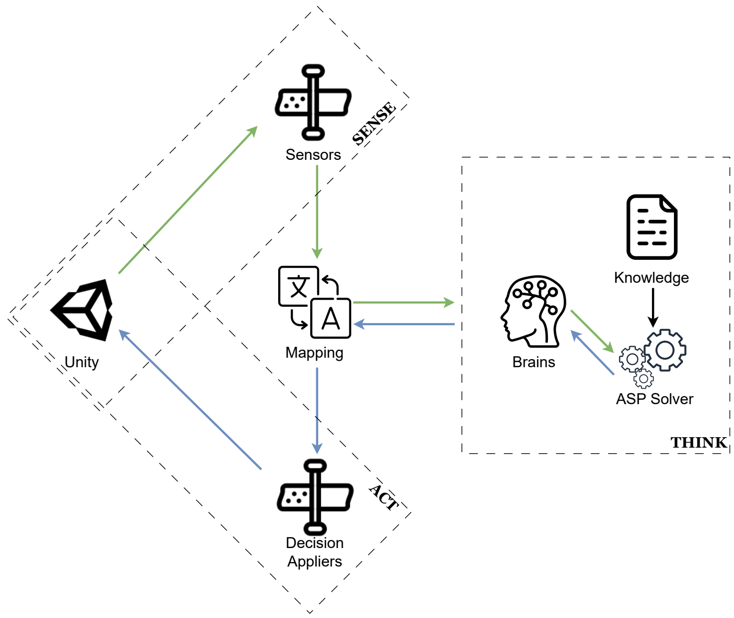


Figure 1: General run-time architecture of the *ThinkEngine* framework

-*It is introduced a proper information passing layer to and from the game loop and the offloaded reasoning jobs*: this includes also data type mapping facilities and an adaptive computational load control for those parts of our *ThinkEngine* working in the main loop. The language of AI modules is not necessarily restricted to ASP, as we provide data type mapping facilities useful to incorporate other declarative languages (e.g., PDDL). Data mapping do not require manual work: no explicit annotation and coding is required.
-*We used ThinkEngine for developing deliberative strategies in a number of videogames available in an online showcase*. The performance of *ThinkEngine* has been assessed on run-time frame rate and on reaction times of artificial players, with reassuring results.

Overview of *ThinkEngine*. At design-time, Unity allows to work on Game Objects (GOs), whereas the game logic can be defined by attaching scripted code to specific game events. In this context, one can use our *ThinkEngine* by adding, wiring and programming *brains* containing declarative specifications written in ASP.

Concerning run-time, *ThinkEngine* interacts in the game scene at hand according to the architecture shown in Figure 1. During the game, Unity iteratively updates the game scene in the canonical *game loop*: *ThinkEngine* makes use of some code parts working in the Unity main loop, but it also uses additional threads for offloading decision tasks. Intuitively, one or more programmable brains (i.e. modules embedding ASP logic programs) are in charge of decision-making tasks and run in separate *ThinkEngine* threads. Brains are connected to the game scene using *sensors*, *actuators* and *plans*. Sensors allow read access to desired parts of the current game state, whereas actuators and plans allow to make changes to the game scene itself.

Reactive brains operate immediately on actuators, while *planning brains* make possible to declaratively program more complex, deliberative-like plans. A plan P is a list of actions $[a_1 \ldots, a_n]$ to be executed in the given sequence. One or more planning brains, each of which capable of generating a plan, can be associated to a given GO. Plans are associated to a priority value and, once generated, they are submitted for execution to a scheduler. A designer can decide how and when to abort a plan and how and when to trigger re-planning. Re-planning can produce new plans which replace older plan versions, so to cope with game state changes.

Planning brains are grouped by GO. Each group has its own *planning scheduler*. A planning scheduler PS has a current plan R and manages a queue of available plans $Q = [P_1, \ldots, P_m]$, where subscripts denote priority (lower subscript values denote higher priority). PS periodically checks if R has to be interrupted and, possibly, substituted with a different plan in Q.

Performance. We tested *ThinkEngine* in several classic games like *Space Invaders*, *Pacman* and *Frogger*, in which we added an automated player whose artificial intelligence is managed via both planning brains and reactive brains. Concerning performance, we considered the following measures: *i*) the frame rate, the most common metric in the videogame field, representing the number of screen updates per second that can be achieved given the computational burden of the game implementation at hand; *ii*) the required number of frames to achieve a complete sensors update cycle; this depends on the number of sensors in the scene; *iii*) the required time to compute a new plan; this affects the delay between the event triggering the generation of a plan and the execution of the first action of the plan. Frame rate was not affected by the introduction of the *ThinkEngine*. Concerning the sensor update cycle, the higher the number of sensors in the scene the more frames the update cycle coroutine is spread on, resulting in some reasonable delay of the starting of the reasoning task. As for the required time to compute a new plan, it is clear that plan size should be kept under a certain limit. In many scenarios, the usage of an incremental solver drastically reduces evaluation times.

Final remarks. To the best of our knowledge, our contribution is the first attempt at introducing declarative methods in the general setting of commercial game engines. As a distinctive feature, the *ThinkEngine* proposes a hybrid architecture in which procedural and declarative parts coexists, and where integration and computational offload issues are explicitly addressed. In future work we aim to improve the integration level of *ThinkEngine* since using declarative paradigms like ASP can be not so natural for game developers. Concerning the real-time performance problem, improvement has been gained by adding an incremental ASP solver, while the sensor update cycle has been optimized drastically lowering the number of frames required to perform the update [1] . Our *ThinkEngine* is publicly available alongside a games showcase.[1].

References

[1] D. Angilica, G.M.D. Giorgio & G. Ianni (2023): *On the impact of sensors update in declarative AI for videogames*. In: ASPOCP 2023, CEUR Workshop Proceedings 3437, CEUR-WS.org.

[2] F. Calimeri et al. (2016): *Angry-HEX: An Artificial Player for Angry Birds Based on Declarative Knowledge Bases*. IEEE TCIAIG 8(2), pp. 128–139, doi:10.1109/TCIAIG.2015.2509600.

[3] G. Ianni et al. (2020): *Incremental maintenance of overgrounded logic programs with tailored simplifications*. Theory Pract. Log. Program., doi:10.1017/S147106842000040X.

[4] J. McCarthy (1999): *Elaboration Tolerance*. Keynote speech at CommonSense 1998. Available at `http://www-formal.stanford.edu/jmc/elaboration.html`.

[5] R.R. Murphy (2000): *Introduction to AI Robotics*, chapter 7, pp. 257–292. MIT Press. Chapter 7: The Hybrid Deliberative/Reactive paradigm.

[6] A.M. Smith & M. Mateas (2011): *Answer Set Programming for Procedural Content Generation: A Design Space Approach*. IEEE TCIAIG 3(3), pp. 187–200, doi:10.1109/TCIAIG.2011.2158545.

[1] `https://github.com/DeMaCS-UNICAL/ThinkEngine` and `https://github.com/DeMaCS-UNICAL/ThinkEngine-Showcase`

stableKanren: Integrating Stable Model Semantics with miniKanren (Extended Abstract)

Xiangyu Guo James Smith Ajay Bansal
School of Computing and Augmented Intelligence
Arizona State University, Tempe, Arizona
Xiangyu.Guo@asu.edu jsmit106@asu.edu Ajay.Bansal@asu.edu

miniKanren implements the essence of Prolog and allows easy access and modifications of underlying resolution and unification. The resolution stream constructed by miniKanren is static. We realize that a dynamic resolution stream is needed to support non-monotonic reasoning. So, we extend the traditional resolution and unification to achieve our goal. We present stableKanren, a miniKanren extension with normal logic program support under stable model semantics. stableKanren adds multiple innovative macros to compile a normal logic program into a program with its complement form, obtain the domain of a variable under different contexts, and generate a new stream during resolution.

1 Overview

miniKanren is a family of languages specially designed for relational programming. The core miniKanren implementation is purely functional and is designed to be easily modified and extended [1]. Unlike Prolog, miniKanren uses a complete *interleaving search* strategy on a stream to simulate backtracking. Unlike Mercury [9], miniKanren uses full unification, which is required to implement goals that take only fresh logic variables as their arguments. Unlike Curry [6], miniKanren did not use residuation to suspend execution on non-ground terms. The core implementation was hosted on a functional language, *Scheme* [2] and introduces only a few operators to users: $==$ for *unification*, *fresh* for *existential quantification*, *conde* for *disjunction*, and a *run* interface.

To the best of our knowledge, only a few attempts have been made to add negation to miniKanren. The constructive negation used by Moiseenko [8] works for stratified programs only. Also, the semantics of negation is more like a filter, where the solver executes the positive goal inside the negation. It gets a differential set between the substitutions before and after the negated goal and eventually subtracts the differential set from the original set. However, none of the functional language-based implementations supported negation in a non-stratified normal program. By integrating the stable model semantics into miniKanren, we believe we have given it a more proper or widely accepted negation semantics.

Unlike a bottom-up solver that grounds all variables and propagates the constraint among values, a top-down solver, such as Prolog and miniKanren, uses resolution and unification to obtain a minimal model of the program. The advantage of top-down solving is that it produces answers without issues with blow-ups. Gupta et al. introduce coinductive logic (co-LP) and co-SLD resolution to handle the infinite terms in logic programs with co-SLD produces the greatest fixed point of a program [5]. Later, Min et al. evolved co-SLD to co-SLDNF to achieve normal program solving [7]. However, we believe that sticking with the least fixed point of SLD resolution is closer to stable model semantics, and the transition is simpler. Therefore, unlike co-SLD and co-SLDNF, our algorithms still produce the least fixed points as the original SLD resolution and focus on adding stable model semantics under finite Herbrand models.

miniKanren takes advantage of the traits (macros and continuations) of Scheme in its implementation. They use macros to transform the input program into a continuations-based search stream that works

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 298–300, doi:10.4204/EPTCS.416.28

perfectly for monotonic reasoning. The static stream does not change once it is compiled (transformed). However, for non-monotonic reasoning, as in normal logic programs, the information we obtain during the run-time will change the outcome of a previous search, such as reverting a result, adjusting a conflict, or generating different results. Therefore, a dynamic search stream is needed. Our extension, stableKanren [4], generates or modifies the stream (code) on the fly based on the new information received during the resolution. Hence, we advance macros and continuations further into stable model solving. We use macros and continuations to form a static search stream and to modify a stream to weave the run-time information into the streams dynamically.

From the stable model semantics definition, we have the following relationship between the stable model of Π and the minimal model of the reduct Π^M.

Definition 1.1 (stable model property) *Let Π be a propositional program and M be an interpretation. M is a stable model of Π if M is a minimal model of the reduct Π^M.*

Stable model semantics consider a minimal model of the corresponding reduct program to be one of the program's stable models. We want our algorithms to simultaneously produce the minimal model and its corresponding reduct program. The underlying resolution process guarantees the production of a minimal model as long as the input program can be handled. The reduct created from the interpretation removes the recursive rules and the negation completely so that the program can be handled using traditional Prolog to produce the minimal model ([3], Remark 2). Then, the interpretation must be verified to show that it is the minimal model of the reduct program.

Initially, the resolution had only one role: selecting a goal. Unification has two roles: the first is to assign a truth value to a goal, and the second is to assign a value to a variable. A goal is proven to be true iff all of its variables are successfully unified, and a goal is grounded iff all of its variables are unified with a value. Both resolution and unification require changes to support the recursive rules and negation introduced by the normal program. We grant more roles to resolution and unification. For resolution, it has four more roles: distributing negation to the unification level, producing truth value for the recursive rules, getting the domain of a variable, and continuing execution after getting a partial result. Unification has to produce the truth value for negated unification.

To prove a rule (statement) with a head H to be true, the resolution tries to prove each sub-goal individually in the rule's body. To prove $\overline{H}$ to be true, we know H is failing somewhere in the rule's body among one of the sub-goals. Therefore, we need to distribute negation to the unification level. For example, consider the logic statement;

$$H(X,Y) \leftarrow \exists X,Y(B_1(X,Y) \wedge B_2(X) \wedge B_3(Y))$$

We assume that under negation $\neg H(X,Y)$, the variables X and Y are safe, always with a value x and y. For this reason, we safely drop the $\exists$ quantifier and focus on the rule's body transformation with assigned values. We obtain the propositional form of the rule as follows;

$$\neg H^{x,y} \stackrel{?}{\leftarrow} transform(B_1^{x,y} \wedge B_2^x \wedge B_3^y)$$

Each sub-goal in a rule's body could fail, and when a sub-goal fails, the prior sub-goals must have succeeded. To capture this property, its transformation should be a disjunction of the negation to each sub-goal in conjunction with all sub-goals before the current one. Furthermore, we noticed that each sub-goal works as a checker, and the values are checked by the unification independently inside the sub-goal could fail. Therefore, the transformation applies to the sub-goal and each variable to distribute the negation to the unification level.

$$\neg H \leftarrow \neg B_1^x \vee (B_1^x \wedge \neg B_1^y) \vee (B_1^x \wedge B_1^y \wedge \neg B_2^x) \vee (B_1^x \wedge B_1^y \wedge B_2^x \wedge \neg B_3^y))$$

stableKanren supports normal logic programs and solves the n-queens problem faster than s(ASP) in a top-down fashion. We did performance testing on a 2012 MacBook Pro with 2.6GHz Intel Quad-Cores i7, 8GB memory, and macOS 10.15.7. We set an hour timeout unless the number of queens is less than or equal to 8. Table 1 shows the testing result time in seconds. The prefix “1” means finding one answer, and “all” means finding all answers.

Table 1: Time on finding one or all nqueens solutions in stableKanren and s(ASP).

nqueens	#solutions	1-s(ASP)	1-stableKanren	all-s(ASP)	all-stableKanren
1	1	0.15	0	0.17	0
2	0	1.65	0	1.66	0
3	0	9.38	0	9.57	0
4	2	24.20	0	58.81	0
5	10	23.69	0.01	397.31	0.01
6	4	421.88	0.03	2470.24	0.08
7	40	148.27	0.04	13852.54	0.43
8	92	4649.88	0.36	131429.13	4.08

References

[1] Daniel P. Friedman, William E. Byrd & Oleg Kiselyov (2005): *The Reasoned Schemer*. The MIT Press, MIT Press, doi:10.7551/mitpress/5801.001.0001.

[2] Daniel P. Friedman & Matthias Felleisen (1996): *The Little Schemer (4th Ed.)*. MIT Press, Cambridge, MA, USA, doi:10.1016/0898-1221(96)87329-9.

[3] Michael Gelfond & Vladimir Lifschitz (1988): *The stable model semantics for logic programming*. In: *ICLP/SLP*, 88, Cambridge, MA, pp. 1070–1080.

[4] Xiangyu Guo, James Smith & Ajay Bansal (2023): *StableKanren: Integrating Stable Model Semantics with MiniKanren*. In: *Proceedings of the 25th International Symposium on Principles and Practice of Declarative Programming*, PPDP ’23, Association for Computing Machinery, New York, NY, USA, doi:10.1145/3610612.3610617.

[5] Gopal Gupta, Ajay Bansal, Richard Min, Luke Simon & Ajay Mallya (2007): *Coinductive Logic Programming and Its Applications*. In: *Proceedings of the 23rd International Conference on Logic Programming*, ICLP’07, Springer-Verlag, Berlin, Heidelberg, pp. 27–44, doi:10.1007/978-3-540-74610-2_4.

[6] Michael Hanus, Herbert Kuchen & Juan Jose Moreno-Navarro (1995): *Curry: A truly functional logic language*. In: *Proc. ILPS*, 95, pp. 95–107.

[7] Richard Min & Gopal Gupta (2010): *Coinductive Logic Programming with Negation*. In Danny De Schreye, editor: *Logic-Based Program Synthesis and Transformation*, Springer Berlin Heidelberg, pp. 97–112, doi:10.1007/978-3-642-12592-8_8.

[8] Evgenii Moiseenko (2019): *Constructive negation for miniKanren*. In: *ICFP 2019, The miniKanren and Relational Programming Workshop*.

[9] 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. High-Performance Implementations of Logic Programming Systems.

Alda: Integrating Logic Rules with Everything Else, Seamlessly (System Demonstration)*

Yanhong A. Liu Scott D. Stoller Yi Tong Bo Lin
Computer Science Department, Stony Brook University, Stony Brook, NY
{liu,stoller,yittong}@cs.stonybrook.edu

Sets and rules have been used for easier programming since the late 1960s. While sets are central to database programming with SQL and are also supported as built-ins in high-level languages like Python, logic rules have been supported as libraries or in rule-based languages with limited extensions for other features. However, rules are central to deductive database and knowledge base programming, and better support is needed.

This system demonstration highlights the design of a powerful language, Alda [16, 14], that supports logic rules together with sets, functions, updates, and objects, all as seamlessly integrated built-ins, including concurrent and distributed processes. The key idea is to allow sets of rules to be defined in any scope, support predicates in rules as set-valued variables that can be used and updated directly, and support queries using rules as either explicit or implicit automatic calls to an inference function.

Alda has a formal semantics [15] and is implemented by building on an object-oriented language (DistAlgo [13, 3] extending Python [18]) and an efficient logic rule system (XSB [19, 20]). It has been used successfully on benchmarks and problems from a wide variety of application domains—including those in OpenRuleBench [6], role-based access control (RBAC) [1, 4], and program analysis—with generally good performance [17]. Our implementation and benchmarks are publicly available [23].

This system demonstration shows how Alda is used for OpenRuleBench benchmarks, ANSI standard for role-based access control, and program analysis for large Python programs, including with persistence support for large datasets, all programmed seamlessly without boiler-plate code. For comparisons with related work on rule languages and benchmarking, see [16, 14, 17].

An example

Figure 1 shows an example program in Alda. It is for a small portion of the ANSI standard for role-based access control (RBAC) [1, 4]. It shows the uses (with line numbers in parentheses) of

- classes (1-8, 9-21) with inheritance (9, 11), and object creation (22) with setup (2-3, 10-12);
- sets, including relations (3, 12);
- methods, including procedures (5-6, 13-14) and functions (7-8, 18-19, 20-21), and calls (23, 24);
- updates, including initialization (3, 12) and membership changes (6, 14); and
- queries, including set queries (8, 19 after union "+", 21) and queries using rules (19 before "+");

where the rules are defined in a rule set (15-17), explained in the next part.

Note that queries using set comprehensions (e.g., on lines 8, 19, 21) can also be expressed by using rules and inference, even though comprehensions are more widely used. However, only some queries

*This work was supported in part by NSF under grants CCF-1954837, CCF-1414078, and IIS-1447549 and ONR under grants N00014-21-1-2719, N00014-20-1-2751, and N00014-15-1-2208.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 301–305, doi:10.4204/EPTCS.416.29

```
1  class CoreRBAC:                    # class for Core RBAC component/object
2    def setup():                     # method to set up the object, with no arguments
3      self.USERS, self.ROLES, self.UR := {},{},{}
4                                     # set users, roles, user-role pairs to empty sets
5    def AddRole(role):               # method to add a role
6      ROLES.add(role)                # add the role to ROLES
7    def AssignedUsers(role):         # method to return assigned users of a role
8      return {u: u in USERS | (u,role) in UR}  # return set of users having the role
   ...

9  class HierRBAC extends CoreRBAC: # Hierarchical RBAC extending Core RBAC
10   def setup():
11     super().setup()                # call setup of CoreRBAC, to set sets as in there
12     self.RH := {}                  # set ascendant-descendant role pairs to empty set
13   def AddInheritance(a,d):         # to add inherit. of an ascendant by a descendant
14     RH.add((a,d))                  # add pair (a,d) to RH
15   rules trans_rs:                  # rule set defining transitive closure
16     path(x,y) if edge(x,y)         # path holds for (x,y) if edge holds for (x,y)
17     path(x,y) if edge(x,z), path(z,y)  # ... if edge holds for (x,z) and for (z,y)
18   def transRH():                   # to return transitive RH and reflexive role pairs
19     return infer(path, edge=RH, rules=trans_rs) + {(r,r): r in ROLES}
20   def AuthorizedUsers(role):       # to return users having a role transitively
21     return {u: u in USERS, r in ROLES | (u,r) in UR and (r,role) in transRH()}
   ...

22 h = new(HierRBAC, [])              # create HierRBAC object h, with no args to setup
23 h.AddRole('chair')                 # call AddRole of h with role 'chair'
   ...
24 h.AuthorizedUsers('chair')         # call AuthorizedUsers of h with role 'chair'
   ...
```

Figure 1: An example program in Alda, for Role-Based Access Control (RBAC). In rules `trans_rs`, the first rule says there is a path from `x` to `y` if there is an edge from `x` to `y`, and the second rule says there is a path from `x` to `y` if there is an edge from `x` to `z` and there is an edge from `z` to `y`. The call to infer queries and returns the set of pairs for which `path` holds given that `edge` holds for exactly the pairs in set `RH`, by doing inference using rules in `trans_rs`.

using rules and inference can be expressed by using comprehensions; queries using recursive rules (e.g., on lines 16-17) cannot be expressed using comprehensions.

Rules with sets, functions, updates, and objects

In Alda, rules are defined in rule sets, each with a name and optional declarations for the predicates in the rules.

$$\begin{array}{rl} \textit{ruleset} & ::= \texttt{rules}\ \textit{name}\ (\textit{declarations})\texttt{:}\ \textit{rule}+ \\ \textit{rule} & ::= p(\textit{arg}_1\texttt{,}\ \ldots\texttt{,}\ \textit{arg}_a)\ \texttt{if}\ p_1(\textit{arg}_{11}\texttt{,}\ \ldots\texttt{,}\ \textit{arg}_{1a_1})\texttt{,}\ \ldots\texttt{,}\ p_k(\textit{arg}_{k1}\texttt{,}\ \ldots\texttt{,}\ \textit{arg}_{ka_k}) \end{array}$$

In the rule form, p, p_1, ..., p_k denote predicates, $p(arg_1, ..., arg_a)$ denotes that p holds for its tuple of arguments, and if denotes that its left-side conclusion holds if its right-side conditions all hold. In a rule set, predicates not in any conclusion are called base predicates; the other predicates are called derived predicates.

The key ideas of seamless integration of rules with sets, functions, updates, and objects are:

1. a predicate is a set-valued variable that holds the set of tuples for which the predicate is true;
2. queries using rules are calls to an inference function, infer, that computes desired values of derived predicates using given values of base predicates;
3. values of base predicates can be updated directly as for other variables, whereas values of derived predicates can only be updated by infer; and
4. predicates and rule sets can be object attributes as well as global and local names, just as variables and functions can.

Declarative semantics of rules are ensured by automatically maintaining values of derived predicates when values of base predicates are updated, by appropriate implicit calls to infer.

For example, in Figure 1, one could use an object field `transRH` in place of calls to `transRH()` in `AuthorizedUsers(role)`, use the following rule set instead of `trans_rs`, and remove `transRH()`.

```
rules transRH_rs:                # no need to call infer explicitly
  transRH(x,y) if RH(x,y)
  transRH(x,y) if RH(x,z), transRH(z,y)
  transRH(x,x) if ROLES(x)
```

Field `transRH` is automatically maintained at updates to `RH` and `ROLES` by implicit calls to infer.

Higher-order, patterns, distributed programming, and more

Higher-order. Note that predicates in rules as set-valued variables, e.g., `edge`, and calling infer to take or return values of set variables, e.g., `RH` in `edge=RH`, avoids the need of high-order predicates or other sophisticated features, e.g., [2], to reuse rules for different predicates in logic languages.

Patterns. Alda also supports tuple patterns for set elements in set queries (as in DistAlgo [13]) and in queries using rules, e.g., `(1,=x,y) in p` matches any triple in set `p` whose first element is 1 and whose second element equals the value of `x`, and binds `y` to the third element if such a triple exists.

Distributed programming. Of course, by building on DistAlgo, Alda also supports distributed programming with distributed processes, message passing, and high-level queries of message histories, e.g., for distributed RBAC [7, 9], also called trust management [5], in decentralized systems.

Declarations for predicates in rules. Declarations in rules could specify predicate types and scopes, but are designed more importantly for specifying assumptions about predicates being certain, complete, closed, or not [10, 11, 12]. This is to give respective desired semantics for rules with unrestricted negation, quantification, and aggregation.

Python syntax. Note that the examples discussed use an ideal syntax, while the Alda implementation supports the Python syntax. For example, `x := {}` is written as `x = set()` in Python syntax.

Implementation. The Alda implementation compiles rule sets in rules and queries using infer to XSB rules and queries, and compiles the rest to Python, which calls XSB to do the inference. The current implementation supports primarily Datalog rules, but also handles unrestricted negation by using XSB's computation of the well-founded semantics [24]. More general forms of rules and queries can be compiled to rules and queries in XSB or other rule systems using the same approach. In general, any efficient inference algorithm and implementation method can be used to compute the semantics of rules and infer.

Future work. Future work includes (1) support for easy use of different desired semantics, especially with modular use of rules, similar to knowledge units in DA-logic [11]; and (2) efficient implementation with complexity guarantees [8, 21, 22] for computing different desired semantics.

References

[1] ANSI INCITS (2004): *Role-Based Access Control*. ANSI INCITS 359-2004, American National Standards Institute, International Committee for Information Technology Standards.

[2] Weidong Chen, Michael Kifer & David S. Warren (1993): *HiLog: A Foundation for Higher-Order Logic Programming*. Journal of Logic Programming 15(3), pp. 187–230, doi:10.1016/0743-1066(93)90039-J.

[3] (2024): *DistAlgo*. http://distalgo.cs.stonybrook.edu. Accessed July 8, 2024.

[4] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn & Ramaswamy Chandramouli (2001): *Proposed NIST Standard for Role-Based Access Control*. ACM Transactions on Information and Systems Security 4(3), pp. 224–274, doi:10.1145/501978.501980.

[5] Ninghui Li, John C. Mitchell & William H. Winsborough (2002): *Design of a Role-Based Trust-Management Framework*. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy, pp. 114–130, doi:10.1109/SECPRI.2002.1004366.

[6] Senlin Liang, Paul Fodor, Hui Wan & Michael Kifer (2009): *OpenRuleBench: An Analysis of the Performance of Rule Engines*. In: Proceedings of the 18th International Conference on World Wide Web, ACM Press, pp. 601–610, doi:10.1145/1526709.1526790.

[7] Yanhong A. Liu (2018): *Role-Based Access Control as a Programming Challenge*. In: LPOP: Challenges and Advances in Logic and Practice of Programming, https://arxiv.org/abs/2008.07901, pp. 14–17, doi:10.48550/arXiv.2008.07901.

[8] Yanhong A. Liu & Scott D. Stoller (2009): *From Datalog Rules to Efficient Programs with Time and Space Guarantees*. ACM Transactions on Programming Languages and Systems 31(6), pp. 1–38, doi:10.1145/1552309.1552311.

[9] Yanhong A. Liu & Scott D. Stoller (2018): *Easier Rules and Constraints for Programming*. In: LPOP: Challenges and Advances in Logic and Practice of Programming, https://arxiv.org/abs/2008.07901, pp. 52–60, doi:10.48550/arXiv.2008.07901.

[10] Yanhong A. Liu & Scott D. Stoller (2020): *Founded Semantics and Constraint Semantics of Logic Rules*. Journal of Logic and Computation 30(8), pp. 1609–1638, doi:10.1093/logcom/exaa056. Also http://arxiv.org/abs/1606.06269.

[11] Yanhong A. Liu & Scott D. Stoller (2021): *Knowledge of Uncertain Worlds: Programming with Logical Constraints*. Journal of Logic and Computation 31(1), pp. 193–212, doi:10.1093/logcom/exaa077. Also https://arxiv.org/abs/1910.10346.

[12] Yanhong A. Liu & Scott D. Stoller (2022): *Recursive Rules with Aggregation: A Simple Unified Semantics*. Journal of Logic and Computation 32(8), pp. 1659–1693, doi:10.1093/logcom/exac072. Also http://arxiv.org/abs/2007.13053.

[13] Yanhong A. Liu, Scott D. Stoller & Bo Lin (2017): *From Clarity to Efficiency for Distributed Algorithms*. ACM Transactions on Programming Languages and Systems 39(3), pp. 12:1–12:41, doi:10.1145/2994595.

[14] Yanhong A. Liu, Scott D. Stoller, Yi Tong & Bo Lin (2023): *Integrating logic rules with everything else, seamlessly*. Theory and Practice of Logic Programming 23(4), pp. 678–695, doi:10.1017/S1471068423000108.

[15] Yanhong A. Liu, Scott D. Stoller, Yi Tong & Bo Lin (2023): *Integrating logic rules with everything else, seamlessly*. Computing Research Repository arXiv:2305.19202 [cs.PL], doi:10.48550/arXiv.2305.19202.

[16] Yanhong A. Liu, Scott D. Stoller, Yi Tong, Bo Lin & K. Tuncay Tekle (2022): *Programming with Rules and Everything Else, Seamlessly*. Computing Research Repository arXiv:2205.15204 [cs.PL], doi:10.48550/arXiv.2205.15204.

[17] Yanhong A. Liu, Scott D. Stoller, Yi Tong & K. Tuncay Tekle (2023): *Benchmarking for Integrating Logic Rules with Everything Else*. In: Proceedings of the 39th International Conference on Logic Programming (Technical Communications), Open Publishing Association, pp. 12–26, doi:10.4204/EPTCS.385.3.

[18] Python Software Foundation (2024): *Python*. http://python.org/.

[19] Terrance Swift & David S Warren (2012): *XSB: Extending Prolog with tabled logic programming*. Theory and Practice of Logic Programming 12(1-2), pp. 157–187, doi:10.1017/S1471068411000500.

[20] Theresa Swift, David S. Warren, Konstantinos Sagonas, Juliana Freire, Prasad Rao, Baoqiu Cui, Ernie Johnson, Luis de Castro, Rui F. Marques, Diptikalyan Saha, Steve Dawson & Michael Kifer (2022): *The XSB System Version 5.0,x*. http://xsb.sourceforge.net. Latest release May 12, 2022.

[21] K. Tuncay Tekle & Yanhong A. Liu (2010): *Precise Complexity Analysis for Efficient Datalog Queries*. In: Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming, pp. 35–44, doi:10.1145/1836089.1836094.

[22] K. Tuncay Tekle & Yanhong A. Liu (2011): *More Efficient Datalog Queries: Subsumptive Tabling Beats Magic Sets*. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data, pp. 661–672, doi:10.1145/1989323.1989393.

[23] Yi Tong, Bo Lin, Yanhong A. Liu & Scott D. Stoller (2018 (Latest update May, 2024)): *Alda*. http://github.com/DistAlgo/alda.

[24] Allen Van Gelder, Kenneth Ross & John S. Schlipf (1991): *The Well-Founded Semantics for General Logic Programs*. Journal of the ACM 38(3), pp. 620–650, doi:10.1145/116825.116838.

Generating Causally Compliant Counterfactual Explanations using ASP*

Sopam Dasgupta
Department of Computer Science
The University of Texas at Dallas
Texas, USA
sopam.dasgupta@utdallas.edu

This research is focused on generating achievable counterfactual explanations. Given a negative outcome computed by a machine learning model or a decision system, the novel *CoGS* approach generates (i) a counterfactual solution that represents a positive outcome and (ii) a path that will take us from the negative outcome to the positive one, where each node in the path represents a change in an attribute (feature) value. *CoGS* computes paths that respect the causal constraints among features. Thus, the counterfactuals computed by *CoGS* are realistic. *CoGS* utilizes rule-based machine learning algorithms to model causal dependencies between features. The paper discusses the current status of the research and the preliminary results obtained.

1 Introduction

Predictive models used in automated decision-making processes (job-candidate filtering, loan approvals) often function as black boxes, making it difficult to understand their internal reasoning for decision-making. The decisions can have significant consequences, leading individuals to seek satisfactory explanations, especially for an unfavourable (negative) decision. Explaining these decisions presents a significant challenge. Additionally, users want to understand the changes necessary to flip a negative decision into a positive one.

Following Wachter et al.'s [15] approach in this research, counterfactuals are employed to explain a machine learning model's reasoning behind a prediction. Counterfactuals help answer the question: "What changes should be made to input attributes or features to flip a negative outcome to a positive one?" Counterfactuals also serve as a good explanation for a prediction. Wachter et al. [15] use statistical techniques by examining the proximity of points in the N-dimensional feature space to find counterfactuals. This paper presents the *Counterfactual Generation with s(CASP) (CoGS)* framework, which generates counterfactual explanations from *rule-based machine learning (RBML)* algorithms such as FOLD-SE [16]. *CoGS* makes two advances compared to Wachter et al.'s work: (i) It computes counterfactuals using *RBML* algorithms and ASP [8] rather than statistical techniques, and (ii) It considers causal dependencies among features when computing these counterfactuals. Another novelty of the *CoGS* framework is that it further leverages the FOLD-SE algorithm [16] to automatically discover potential dependencies between features that a user subsequently approves.

CoGS models various scenarios (or worlds): the current *initial state i* represents a negative outcome, and the *goal state g* represents a positive outcome. A state is represented as a set of feature-value pairs. *CoGS* finds a path from the *initial state i* to the *goal state g* by performing interventions (or transitions), where each intervention corresponds to changing a feature value while considering causal

*Authors supported by US NSF Grants IIS 1910131, US DoD, and industry grants.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 306–313, doi:10.4204/EPTCS.416.30

dependencies among features. These interventions ensure realistic and achievable changes that will take us from state i to g. *CoGS* relies on common-sense reasoning, implemented through answer set programming (ASP) [8], explicitly using the goal-directed s(CASP) ASP system [1]. The problem of finding these interventions can be viewed as a planning problem [8], except that, unlike the planning problem, the moves (interventions) that take us from one state to another are not mutually independent.

2 Background

Counterfactual Reasoning: Counterfactual reasoning is critical for explaining decisions in machine learning, offering insights on achieving desired outcomes by imagining plausible alternate scenarios. Wachter et al. [15] advocated using counterfactual explanations to explain individual decisions, suggesting what changes could flip a negative outcome to a positive one. However, this approach often ignored causal dependencies, leading to unrealistic suggestions. For a binary classifier given by $f : X \to \{0,1\}$, we define a set of counterfactual explanations $\hat{x}$ for a factual input $x \in X$ as $CF_f(x) = \{\hat{x} \in X | f(x) \neq f(\hat{x})\}$. This set includes all inputs $\hat{x}$ leading to different predictions than the original input x under f.

Causality Considerations: Causality relates to cause-effect relationship among predicates. P is the cause of Q, if $(P \Rightarrow Q) \wedge (\neg P \Rightarrow \neg Q)$ [13]. We say that Q is causally dependent on P. Causality is crucial for generating realistic counterfactuals. For example, increasing the *credit score* to be ‘high’ while still being under increasing *debt* obligations is unrealistic due to their causal link. Realistic counterfactuals must model these dependencies to ensure achievable changes.

ASP, s(CASP) Answer Set Programming (ASP) is a paradigm for knowledge representation and reasoning [6, 2, 8]. ASP encodes feature knowledge, decision-making rules and causal rules, enabling the automatic generation of counterfactual explanations using this symbolic knowledge. **s(CASP)** is a goal-directed ASP system that executes answer set programs in a top-down manner without grounding [1]. s(CASP) adopts *program completion*, turning “if” rules $(P \Rightarrow Q)$ into “if and only if” rules $((P \Rightarrow Q) \wedge (\neg P \Rightarrow \neg Q))$ which models causality.

FOLD-SE: FOLD-SE [16], is an efficient *rule-based machine learning (RBML)* algorithm for classification tasks. It generates explainable models and learns causal rules from data. It maintains scalability and accuracy, making it a reliable component for the *CoGS* framework, which leverages these rules for generating counterfactuals.

The Planning Problem: Planning involves finding a sequence of transitions from an initial state to a goal state while adhering to constraints. In ASP, this problem is encoded in a logic program with rules defining transitions and constraints restricting the allowed transitions [8]. Solutions are represented as a series of transitions through intermediate states. Each state is represented as a set of facts or logical predicates. Solving the planning problem involves searching for a path of transitions that meets the goal conditions within the constraints. *CoGS* can be thought of as a framework to find a plan—a series of interventions that change feature values—that will take us from the initial state to the final goal state. However, unlike the planning domain, the interventions (moves) are not independent of each other due to causal dependencies among features.

3 Research Goal

This research aims to develop a framework that can encode feature knowledge, decision-making rules, and causal rules, enabling the automatic generation of counterfactual explanations using symbolic knowl-

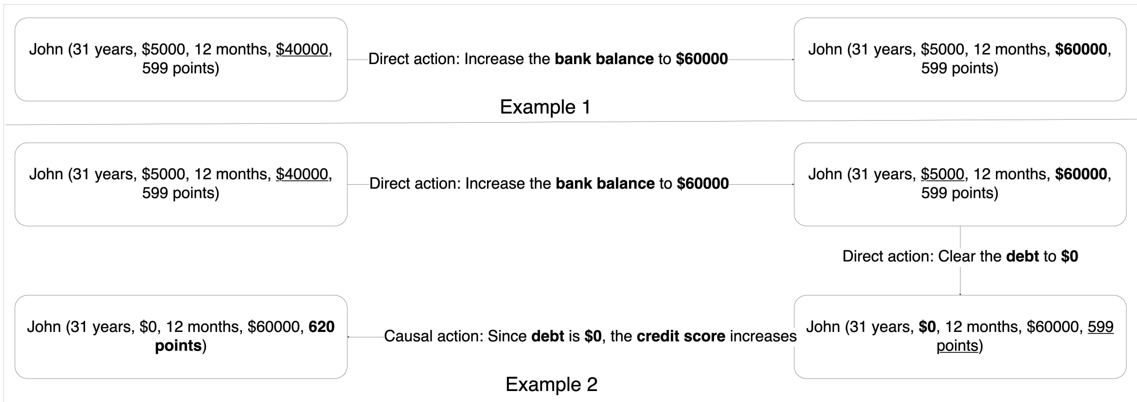


Figure 1: **Top:** Example 1 shows how John goes from being rejected for a loan to having his loan approved. Here the bank only considers the *bank balance* for loan approval. John does a direct action to increase his *bank balance* to \$60000. **Bottom:** Example 2 shows how John goes from being rejected for a loan to having his loan approved. Here the bank considers both *bank balance* as well as *credit score* for loan approval. While the *bank balance* is directly altered by John, altering the *credit score* requires John to directly alter his *debt* obligations first. After clearing his *debt*, the causal effect of having \$0 *debt* increases John's *credit score* to 620 *point*. This is the causal action

edge represented by an ASP program. The objective then is to use the ASP program to solve a version of the Planning Problem where the desired *goal state g* is our counterfactual state for solving the task at hand. This research focuses on generating counterfactuals for models that use decision rules (rule-based models). These rules are provided as explanations to justify decisions made by a governing authority, for example, a bank rejecting a loan due to a low *bank balance* or a low *credit score*. However, this can also be translated to statistical models by generating a rule-based approximation of these models.

Currently, most counterfactual-based approaches generate explanations without accurately accounting for the causal dependencies among features. These methods assume that the suggested changes will directly lead to a desired outcome, such as turning a negative decision into a positive one. As a result, these approaches are effective/practical only under two conditions: (1) when the features are independent or (2) when causal dependencies between features are irrelevant because only causally independent features are modified. For instance, consider Example 1 in Figure 1: John's loan application was rejected due to a *bank balance* of $\leq$ \$60000. The counterfactual solution suggests *increasing* his *bank balance* to \$60000. This recommendation is straightforward and achievable, as the *bank balance* can be directly altered without affecting other features.

However, since most of these counterfactual-based approaches do not (accurately) model the causal dependencies between the features, changing certain features results in unintended changes to other features. Take Example 2 in Figure 1: John's loan application was rejected due to his poor ($\leq$ 599 *points*) *credit score*. The counterfactual solution tells him to *increase* his *credit score*. However, the *credit score* is causally linked to the current *debt* obligations and cannot be directly increased. Thus, the counterfactual solution may ultimately prevent the expected positive outcome from being achieved or even result in the generated counterfactual requiring a higher cost than initially assumed.

The proposed solution, *CoGS*, would model the causal dependencies and provide a procedure/path informing in a step-by-step manner on what changes to make to achieve a counterfactual solution realistically. Example 2 assumes John has his loan application rejected due to his poor *credit score*. The *CoGS*

Features	Initial State	Action	Goal State	Time (ms)
Checking account status	$\geq$ 200	N/A	$\geq$ 200	
Credit history	no credits taken/all credits paid back duly	N/A	no credits taken/all credits paid back duly	
Property	real estate	**Direct**	car or other	3236
Duration months	7	N/A	7	
Credit amount	500	N/A	500	
Job	unemployed	N/A	unemployed	
Present Employment Since	unemployed/unskilled-non-resident	N/A	unemployed/unskilled-non-resident	

Table 1: Transitions to goal states for the *German* dataset: The value of *Property* changes from *real estate* to *car or other*.

Features	Initial State	Action	Intermediate	Action	Goal State	Time (ms)
Marital_Status	never_married	N/A	never_married	**Causal**	married_civ_spouse	
Capital Gain	\$6000	N/A	N/A	N/A	> 6849 and ≤ 99999	
Education_num	7	N/A	N/A	N/A	7	1126
Relationship	unmarried	**Direct**	husband	N/A	husband	
Sex	male	N/A	N/A	N/A	male	
Age	28	N/A	N/A	N/A	28	

Table 2: Transitions to goal states for the *Adult* dataset: The value of *Relationship* changes from *unmarried* to *husband*. This has a causal effect of altering *Marital Status* to *married_civ_spouse*.

solution tells him to *clear* his *debt* obligations. This increases John's *credit score*, ultimately approving the loan. We approach this through the lens of the planning problem that provides us with a step-by-step path of the changes to make until we reach the goal state (counterfactual).

To summarize, the research goal is twofold: 1) Given the Decision Rules D that give a negative outcome, we capture the causal dependencies C amongst the features using user-defined rules or rules learnt using *RBML* algorithms, and 2) Solve the planning problem where the *goal state* g is defined as a state that is consistent with the causal rules C and inconsistent with the decision rules D.

4 Preliminary Results

We applied the *CoGS* methodology to rules generated by the FOLD-SE algorithm (code on GitHub [7]). Our experiments use the German dataset [9], the Adult dataset [3], and the Car Evaluation dataset [5]. These are popular datasets in the UCI Machine Learning repository [12]. The German dataset contains demographic data with labels for credit risk ('*good*' or '*bad*'), with records with the label '*good*' vastly outnumbering those labelled '*bad*'. The Adult dataset includes demographic information with labels indicating income ('$=<\$50k/year$' or '$>\$50k/year$'). The Car Evaluation dataset provides information on the acceptability of a used car being purchased. We relabelled the Car Evaluation dataset to *'acceptable'* and *'unacceptable'* to generate the counterfactuals.

For the (imbalanced) German dataset, the learned FOLD-SE rules determine a '*good*' credit rating, with the undesired outcome being a '*good*' rating since the aim is to identify criteria making someone a credit risk ('*bad*' rating). Additionally, causal rules are also learnt using FOLD-SE and verified (for example, if the feature **'Job'** has the value *'unemployed'*, then the feature **'Present employment since'**

Features	Initial State	Action	Goal State	Time (ms)
persons	4	N/A	4	1221
maint	low	**Direct**	medium	
buying	medium	N/A	medium	
safety	medium	N/A	medium	

Table 3: Transitions to goal states for the *Car Evaluation* dataset: The value of *maint* goes from *low* to *medium*.

Dataset	# of Features Used	# of Counterfactuals
Adult	6	112
Cars	4	78
German	7	240

Table 4: Table showing a Number of Counterfactuals produce by the *is_counterfactual* function given all possible states.

should have the value *'unemployed/unskilled-non-resident'*). We learn the rules to verify these assumptions on cause-effect dependencies.

Path to the Counterfactual: By using these rules that identify individuals with a '*good*' rating, we found a path to the counterfactuals, thereby depicting steps to fall from a *'good'* to a *'bad'* rating in Table 1. Similarly, we learn the causal rules and the rules for the undesired outcome for the *Adult* dataset (undesired outcome: '$=<$ \$50$k/year$') as shown in Table 2. For the *Car Evaluation* dataset (undesired outcome: *'unacceptable'*) shown in Table 3, we only learn the rules for the undesired outcome as there are no causal dependencies (FOLD-SE did not generate any either). Tables 1, 2 and 3 show a path to each dataset's counterfactual goal state for a specific instance. Note that the execution time for finding the counterfactuals is also reported. While we have only shown specific paths in Tables 1, 2 and 3, our *CoGS* methodology can generate all possible paths from an original instance to a counterfactual.

Number of Counterfactual Sets: Note that each path may represent a set of counterfactuals. This is because numerical features may range over an interval. Thus, *CoGS* generates 240 sets of counterfactuals for the German dataset, 112 for the Adult dataset, and 78 for the Car Evaluation dataset (Table 4).

5 Related Work

Various methods for generating counterfactual explanations in machine learning have been proposed. Wachter et al. [15] aimed to provide transparency in automated decision-making by suggesting changes individuals could make to achieve desired outcomes. However, they ignored causal dependencies, resulting in unrealistic suggestions. Utsun et al. [14] introduced algorithmic recourse, offering actionable paths to desired outcomes but assuming feature independence, which is often unrealistic. *CoGS* rectifies this by incorporating causal dependencies. Karimi et al. [10] focused on feature immutability and diverse counterfactuals, ensuring features like gender or age are not altered and maintained model-agnosticism. However, this method also assumes feature independence, limiting realism. White et al. [17] showed how counterfactuals can enhance model performance and explanation accuracy. Karimi et al. [11] further emphasized incorporating causal rules in counterfactual generation for realistic and achievable interventions. However, their method did not use the 'if and only' property, which is vital in incorporating the effects of causal dependence. *CoGS* rectified this by utilizing Answer Set Programming (ASP), which does not require grounding as it leverages s(CASP) to generate counterfactual explanations, providing a clear path from undesired to desired outcomes.

Bertossi [4] utilizes Answer Set Programming (ASP) to generate *causal explanations* by identifying *minimal cardinality sets* using counterfactuals. These *minimal cardinality sets* are used to compute scores to identify causal explanations. Unlike their work, *CoGS* focuses on defining the causal dependencies amongst features and incorporating them into the framework. As a result of this *CoGS* returns a series of steps to take to go from an original instance to a counterfactual instance which accounts for the causal impact of making interventions when going from one state to another.

The main contribution of this paper is the Counterfactual Generation with s(CASP) (*CoGS*) framework for automatically generating counterfactuals while taking causal dependencies into account to flip a negative outcome to a positive one. *CoGS* has the ability to find minimal paths by iteratively adjusting the path length. This ensures that explanations are both minimal and causally consistent. *CoGS* is flexible, generating counterfactuals irrespective of the underlying rule-based machine learning (*RBML*) algorithm. The causal dependencies can be learned from data using any *RBML* algorithm, such as FOLD-SE. The goal-directed s(CASP) ASP system plays a crucial role, as it allows us to compute a possible world in which a query `Q` fails by finding the world in which the query `not Q` succeeds. *CoGS* advances the state of the art by combining counterfactual reasoning, causal modelling, and ASP-based planning, offering a robust framework for realistic and actionable counterfactual explanations. Our experimental results show that counterfactuals can be computed for complex models in a reasonable amount of time.

6 Limitations and Planned Work

One of the limitations of of *CoGS* is its high computational time, which may lead to scalability issues. We are currently looking for ways to address this problem by replacing the multiple feature-independent values of a given feature with a single placeholder value. The plans for expanding on the work of *CoGS* include:

- Improving the execution time taken to generate counterfactual solutions as well as paths from the current outcome to the counterfactual instance.
- Expanding *CoGS* to generate counterfactuals for statistical machine learning methods: By running an *RBML* algorithm on the predictions of the statistical model, a rule-based model approximation is generated. This approximation can then be used as the decision rules D corresponding to the statistical model that is required by the *CoGS* method.
- Improve the performance of machine learning systems: When machine learning models are trained on imbalanced datasets, the learned model often optimizes its performance on accurately predicting the majority class compared to the minority class. The plan is to generate counterfactual instances of the majority class, which will help us generate instances that belong to the minority class. The expectation is that the machine learning model trained on the modified training data will perform better with respect to both the majority and minority classes versus the original model trained on the original dataset (imbalanced).

7 Conclusion

To conclude, this research is focused on automatically generating counterfactual solutions. This is accomplished by modelling causality and providing a path depicting the series of steps to be taken to achieve the counterfactual solution. This research proposes to do that by modelling the causal relationships that exist between features and the decision rules that led to the undesired negative outcome. Using these rules, a counterfactual solution is obtained. Finally, a version of the planning problem whose *goal state g* is the counterfactual solution, and the *initial state i* is the original negative outcome is solved. The generated plan corresponds to the path representing feature changes that take us from i to g. These rules, as well as the modified planning problem, are modelled in s(CASP), a goal-directed answer set programming system.

References

[1] Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple & Gopal Gupta (2018): *Constraint Answer Set Programming without Grounding*. Theory Pract. Log. Program. 18(3-4), pp. 337–354, doi:10.1017/S1471068418000285.

[2] Chitta Baral (2003): *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, doi:10.1017/CBO9780511543357.

[3] Barry Becker & Ronny Kohavi (1996): *Adult*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.

[4] Leopoldo E. Bertossi (2020): *An ASP-Based Approach to Counterfactual Explanations for Classification*. In Víctor Gutiérrez-Basulto, Tomás Kliegr, Ahmet Soylu, Martin Giese & Dumitru Roman, editors: *Rules and Reasoning - 4th International Joint Conference, RuleML+RR 2020, Oslo, Norway, June 29 - July 1, 2020, Proceedings*, *Lecture Notes in Computer Science* 12173, Springer, pp. 70–81, doi:10.1007/978-3-030-57977-7_5.

[5] Marko Bohanec (1997): *Car Evaluation*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5JP48.

[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] Sopam Dasgupta (2024): *Supplement: CoGS: Causality Constrained Counterfactual Explanations using Goal-directed ASP*. Available at https://github.com/sopam/Supplementary.

[8] M. Gelfond & Y. Kahl (2014): *Knowledge representation, reasoning, and the design of intelligent agents: Answer Set Programming approach*. Cambridge Univ. Press, doi:10.1017/CBO9781139342124.

[9] Hans Hofmann (1994): *Statlog (German Credit Data)*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5NC77.

[10] Amir-Hossein Karimi, Gilles Barthe, Borja Balle & Isabel Valera (2020): *Model-Agnostic Counterfactual Explanations for Consequential Decisions*. In Silvia Chiappa & Roberto Calandra, editors: *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*, *Proceedings of Machine Learning Research* 108, PMLR, pp. 895–905. Available at http://proceedings.mlr.press/v108/karimi20a.html.

[11] Amir-Hossein Karimi, Bernhard Schölkopf & Isabel Valera (2021): *Algorithmic Recourse: from Counterfactual Explanations to Interventions*. In: *Proc. ACM FAccT*, pp. 353–362, doi:10.1145/3442188.3445899.

[12] Kolby Nottingham Markelle Kelly, Rachel Longjohn: *UCI Machine Learning repository*. Available at https://archive.ics.uci.edu/.

[13] Judea Pearl (2009): *Causal inference in statistics: An overview*. *Statistics Surveys* 3(none), pp. 96 – 146, doi:10.1214/09-SS057.

[14] Berk Ustun, Alexander Spangher & Yang Liu (2019): *Actionable Recourse in Linear Classification*. In danah boyd & Jamie H. Morgenstern, editors: *Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT* 2019, Atlanta, GA, USA, January 29-31, 2019*, ACM, pp. 10–19, doi:10.1145/3287560.3287566.

[15] Sandra Wachter, Brent D. Mittelstadt & Chris Russell (2017): *Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR*. *CoRR* abs/1711.00399. arXiv:`1711.00399`.

[16] Huaduo Wang & Gopal Gupta (2024): *FOLD-SE: An Efficient Rule-Based Machine Learning Algorithm with Scalable Explainability*. In Martin Gebser & Ilya Sergey, editors: *Practical Aspects of Declarative Languages - 26th International Symposium, PADL 2024, London, UK, January 15-16, 2024, Proceedings*, *Lecture Notes in Computer Science* 14512, Springer, pp. 37–53, doi:`10.1007/978-3-031-52038-9_3`.

[17] Adam White & Artur S. d'Avila Garcez (2020): *Measurable Counterfactual Local Explanations for Any Classifier*. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín & Jérôme Lang, editors: *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, *Frontiers in Artificial Intelligence and Applications* 325, IOS Press, pp. 2529–2535, doi:`10.3233/FAIA200387`.

Bridging Deep Learning and Logic Programming for Explainability through ILP

Talissa Dreossi
University of Udine
Udine, Italy
talissa.dreossi@uniud.it

My research explores integrating deep learning and logic programming to set the basis for a new generation of AI systems. By combining neural networks with Inductive Logic Programming (ILP), the goal is to construct systems that make accurate predictions and generate comprehensible rules to validate these predictions. Deep learning models process and analyze complex data, while ILP techniques derive logical rules to prove the network's conclusions. Explainable AI methods, like eXplainable Answer Set Programming (XASP), elucidate the reasoning behind these rules and decisions. The focus is on applying ILP frameworks, specifically ILASP and FastLAS, to enhance explainability in various domains. My test cases span weather prediction, the legal field, and image recognition. In weather forecasting, the system will predict events and provides explanations using FastLAS, with plans to integrate recurrent neural networks in the future. In the legal domain, the research focuses on interpreting vague decisions and assisting legal professionals by encoding Italian legal articles and learning reasoning patterns from Court of Cassation decisions using ILASP. For biological laboratories, we will collaborate with a research group to automate spermatozoa morphology classification for Bull Breeding Soundness Evaluation using YOLO networks and ILP to explain classification outcomes. This hybrid approach aims to bridge the gap between the high performance of deep learning models and the transparency of symbolic reasoning, advancing AI by providing interpretable and trustworthy applications.

1 Introduction

The integration of deep learning and logic programming could be a promising approach to creating more interpretable and robust artificial intelligence systems. My research aims to explore and develop a hybrid framework that leverages the strengths of both paradigms. By combining neural networks with ILP [37, 7], the goal is to construct systems capable of not only making accurate predictions but also generating comprehensible rules that validate these predictions. The proposed framework begins with the application of deep learning models, such as neural networks, to process and analyze complex data. Once the neural network produces an output, ILP techniques will be employed to derive logical rules that substantiate the network's conclusions. We also try to apply ILP directly to the data even if often they are too complex or too wide and this limits the capabilities of ILP systems to learn accurate rules. To further enhance the interpretability and reliability of the system, explainable AI methods in logic programming [21], such as eXplainable Answer Set Programming (XASP) [2], are utilized to elucidate the reasoning behind the derived rules and the decision-making process. I am testing this idea on different fields: weather forecasting, legal judgements and image recognition.

In weather forecasting, accurate predictions are vital for mitigating severe weather impacts, protecting lives, and supporting agriculture, transportation, and disaster management. While traditional methods of weather prediction have greatly benefited from neural networks and deep learning techniques, which offer impressive accuracy, these methods often operate as black boxes, making their predictions hard to

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 314–323, doi:10.4204/EPTCS.416.31

understand. This lack of explainability can undermine user trust, impede validation by domain experts, and hinder model refinement. To address these challenges, the research I am pursuing aims to develop a reliable system capable of predicting weather events while also providing explanations. The idea, that was accepted in LPNMR [18], is to use FastLAS [30] which is able to generate an optimal subset of possible solutions. FastLAS can quickly identify the best-fitting hypotheses, so that we are then able to translate the learned rules in a human readable way, as a meteorologist would do.

Similarly, in the legal domain, AI can address complex issues such as the interpretation of vague legal decisions. In facts, legal argumentation often involves vagueness which originates from semantic indeterminacy even when information is available. This branch of my research, which had a recent publication [12], focuses on that while also trying to construct a tool able to assist judges and lawyers in giving an explanation of the final judgement. In particular, I am using some articles of Italian law to test the outcome. In fact, for instance, according to Italian law, street theft ("furto con strappo") can be classified as either an aggravated form of theft or as robbery ("rapina"), depending on factors such as the presence of violence. To tackle this ambiguity problem, we employ Answer Set Programming (ASP) and ILASP [33]. The relevant articles of the Italian legal code are encoded as a set of ASP rules, while decisions from the Court of Cassation are used to make ILASP learn judges' reasoning patterns.

Finally, AI can be employed to speed up human activities in biological laboratory. Indeed, morphological characteristics of bull spermatozoa are typically assessed visually using bright field microscopy following eosin-nigrosine staining, for the Bull Breeding Soundness Evaluation (BBSE). However, this process is time-consuming and demands experienced personnel to achieve reliable results. Given the increasing adoption of genomic selection schemes for young bulls, whose semen is destined for the artificial insemination industry, there is a growing need for a more standardized technique to analyze semen quality. This need is particularly pressing for evaluating spermatozoa abnormalities that impact semen freezing suitability and fertilizing capacity, which are critical due to the widespread use of frozen-thawed semen. Therefore, I am currently developing an AI system for the automated classification of microscope-acquired images of spermatozoa (the study is at the beginning but we are going to present a poster at the European Federation of Animal Science (EEAP) in September). We will employ neural networks, specifically YOLO networks, which can learn and extract relevant features from complex visual data to perform object detection on spermatozoa. This approach will enable us to classify spermatozoa morphology, identifying normal spermatozoa as well as primary and secondary abnormalities. After this initial phase, the plan is to integrate ILP to learn how to identify different morphological characteristics, thereby providing explanations for why specific spermatozoa are classified as abnormal or not.

The contribution is organized as follows: in Section 2 we briefly introduce the concepts of ILP, focusing on ILASP and FastLAS, and of neural networks, deeply on CNNs, RNN and YOLO networks; an overview of the existing literature is reported in Section 3; Section 4 shows the goal of my research and is followed by Section 5 with an view on the current status and results accomplished of the research.

2 Background

In this section I am going to explain the main concept that concern my research interests.

2.1 Answer Set Programming

Answer Set Programming (ASP) is a declarative programming paradigm born for non-monotonic reasoning and, thanks to the efficiency of ASP solvers, widely used for modeling and solving difficult combinatorial problems. An ASP program is composed by:

- atoms, generally writings of the form $p(a_1,\dots,a_n)$, where p is a predicate symbol and a_i are constant or variable symbols with $i \in [1,n]$. Intuitively, such an atom states that the elements represented by $a_1,\dots,a_n$ enjoy the property denoted by p;
- rules r of the form $H \leftarrow A_1,\dots,A_n,\texttt{not}\,B_1,\dots,\texttt{not}\,B_m$, where H, A_i, and B_j are atoms of the form $p(t_1,\dots,t_n)$ where p is a predicate symbol, t_i are constant or variable symbols, and $n \geq 0$.

A literal is either an atom A or its default negation $\texttt{not}A$, i.e., a *naf-literal*. Rules with empty body are called facts, while rules without a head (corresponding to the case $H = \texttt{false}$) are called constraints or denials. If a rule r has not variable symbols, it is said to be *ground*. A set of ground atoms S is a stable model of a program P if it is the unique minimum model of the reduct P^S of P. The reduct P^S is obtained from P by removing all rules whose body is not satisfied by the atoms in S, and removing all naf-literals from the remaining rules [22]. An ASP solver can determine whether a program P has any stable models and, if so, compute the set $AS(P)$ of all such models. A key property of stable models is that if an atom A belongs to a stable model S, there must be at least one rule r in the ground version of P whose body is satisfied by S and whose head is A. This rule r provides an explanation for the truth of A, meaning that we not only know A is true, but also the reason for its truth even if multiple alternative explanations exist.

The ASP language supports various syntactic extensions. One notable extension introduces controlled forms of non-determinism in programs, such as *choice rules* and *cardinality constraints*. Additionally, *built-in atoms* of the form $E_1\, op\, E_2$ (where E_1 and E_2 are expressions with numerical constants, variables, and arithmetic operators, while $op \in \{<,\leq,=,\neq,>,\geq\}$) can be used in rule bodies to model arithmetic comparisons between numerical expressions. During the grounding phase, the variables in these expressions are replaced by constants, and the expressions are evaluated and replaced by `true`/`false`.

2.2 Inductive Logic Programming

Inductive Logic Programming (ILP) [37] is a subfield of machine learning that focuses on learning logical rules from examples and background knowledge. The objective is to discover a hypothesis (a set of logical rules) that explains the given examples within the context of the provided background knowledge (a set of rules). Various ILP frameworks have been proposed in the literature [32]. My research is focused on *Learning from Answer Sets* (LAS), a state-of-the-art ILP framework designed for learning from noisy examples [32].

LAS [33] is a paradigm for learning answer set programs. It is known that in ASP there can be zero or many answer sets of a program. For this reason we can talk about *brave entailment* ($\models_b$) and *cautious entailment* ($\models_a$): an atom a is bravely entailed by a program P if and only if at least one answer set P contains a, while it is cautiously entailed if every answer set contains it. A learning problem in LAS is called a LAS *task*. Specifically, a LAS task is a tuple $T = \langle B, S_M, E\rangle$, where B is an ASP program known as the *background knowledge*, S_M is a set of rules that form the *hypothesis space*, and E is the set of *examples*. To avoid the explicit introduction of a (huge) hypothesis space, it is defined through a *mode bias*, which consists of a pair of sets of mode declarations $\langle M_h, M_b\rangle$. Here, M_h (head mode declarations) specify which predicates can appear in the head of a rule, and M_b (body mode declarations) specify which predicates can appear in the body of a rule. A mode declaration is a literal whose arguments are either `var(t)` or `const(t)`, where `t` is a type. Informally, a literal is *compatible* with a mode declaration m if it can be constructed by replacing every instance of `var(t)` in m with a variable of type `t` and every `const(t)` in m with a constant of type `t`.

Examples in LAS are based on the notion of a *partial interpretation*. A partial interpretation e_{pi} is a pair of sets of ground atoms $\langle e^{inc}, e^{exc}\rangle$, where e^{inc} and e^{exc} are referred to as the *inclusions* and *exclusions* sets respectively. An interpretation I is said to *extend* a partial interpretation e_{pi} if and only if $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$. Unlike conventional ILP, the LAS framework allows for context-dependent examples. In real world setting, data are *noisy*. Noise can be captured by allowing examples to be weighted with a notion of *penalty*. A *weighted context-dependent partial interpretation* (WCDPI) e is a tuple $\langle e_{id}, e_{pen}, e_{cdpi} \rangle$, where e_{id} is an identifier for e, e_{pen} is a positive integer, called a *penalty*, e_{cdpi} is a context-dependent partial interpretation. A task is called *noisy LAS* task when examples are WCDPIs. Formally, a *Noisy LAS* task is a tuple $T^{noise} = \langle B, M, E \rangle$ where B is an ASP program, S_M is the search space, and E is a finite set of WCDPIs. A hypothesis $H \subseteq S_M$ is an inductive solution of T^{noise} iff $\forall e \in E$, $B \cup H$ accepts e. We denote with $\mathscr{T}^{noise}$ the set of all Noisy LAS tasks. If a hypothesis does not accept an example, it *pays the penalty*, which contributes to the overall *cost* of the hypothesis. A scoring function $\mathscr{S}$ assigns a positive real number as score to any ASP program and noisy task T^{noise}. The goal of a *noisy* LAS tasks is to find an *optimal* hypothesis that minimizes the cost over a given hypothesis space and WCDPI examples. Precisely, a hypothesis H is considered the best answer to a noisy LAS task T^{noise} according to $\mathscr{S}$ if H solves T^{noise} and no other solution H' has a better score than H.

FastLAS [30] is a noisy LAS system. It supports user-defined scoring functions, allowing domain-specific optimisation criteria to be used to bias the search; for example, scoring functions can be used to bias towards the cheapest, least risky, safest or most secure set of rules. Furthermore, continuous data types (such as real numbers) are common in machine learning, but many ILP approaches are unable to deal with such data types (without discretisation). FastLAS supports numeric data types, together with binary comparisons over these types.

On the other hand, ILASP is also a framework designed to learn answer set programs from examples and background knowledge. One of its key features is the ability to handle both noise-free and noisy data, making it robust in real-world applications. It infer new rules forcing positive examples to be extended by at least one answer set, while negative examples have not to be extended by any answer set. ILASP uses a *Conflict-Driven ILP* (CDILP) process [29] and, unlike FastLAS, it is also able to learn recursive hypotheses, enabling the modeling of complex relationships within the data.

2.3 Artificial Neural Networks

Regarding Artificial Neural Networks (ANNs), there are numerous articles and books in the literature from which to retrieve information. Therefore, we will cite just the work of Lecun et al. [34] and the survey of Dong et al. [11]. ANN is a computational model inspired by biological neural networks. It is composed by interconnected neurons, functioning as processing units, and connections with adjustable weights. Through training algorithms, these weights are tuned based on input data, ending in a discriminating function capable of classifying various inputs. The network's structure, as its biological counterpart, evolves during training and recall phases to optimize classification performance. Every network is constructed by layers of neurons. While the basic concept of ANNs involves interconnected layers of neurons that process and learn from data, various specialized architectures have been developed to address specific types of tasks and data structures more effectively. In this section, we will give a brief description of Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and finally You Only Look Once Network (YOLO).

CNN Convolutional Neural Networks (CNNs) are primarily designed for processing structured grid data, such as images. They are particularly effective for image recognition, classification, and segmentation tasks. The convolutional layer is the core building block of a CNN. It applies a set of filters (or *kernels*) to the input image, which slide across the width and height of the input. This operation produces a feature map that highlights the presence of certain features in the input, such as edges, textures, or patterns. During this process, the filter performs element-wise multiplication with the pixel values within the patch.

Recurrent Neural Network Recurrent Neural Networks (RNNs) are particularly well-suited for sequence modeling tasks. Unlike traditional feedforward neural networks, which process inputs independently, RNNs have an internal memory that allows them to capture temporal dependencies in sequential data. Their basic architecture consists of three main components: input (at each time step t, the RNN layer receives an input vector $x^{(t)}$, representing the input at that time step); hidden State (the RNN layer maintains a hidden state vector $h^{(t)}$ that encapsulates the network's memory of past inputs up to time step t); recurrent connection (at each time step, the RNN computes the hidden state $h^{(t)}$ based on the current input $x^{(t)}$ and the previous hidden state $h^{(t-1)}$); output (the RNN layer may produce an output $y^{(t)}$ at each time step).

YOLO You Only Look Once (YOLO) is a state-of-the-art, real-time object detection algorithm introduced in 2015 by Redmon et al. [40]. Unlike traditional object detection methods that use a sliding window approach or region proposal networks, YOLO treats object detection as a single regression problem, directly predicting class probabilities and bounding box coordinates from the entire image in one evaluation. Mathematically, YOLO divides an input image into an $S \times S$ grid. Each grid cell is responsible for predicting B bounding boxes and their corresponding confidence scores. Each bounding box prediction consists of five components: $(x, y, w, h, confidence)$. x, y, w and h correspond to the coordinates and dimensions of the box, while confidence score [25] indicates how likely the box contains an object and how accurate the box is.

3 Related Works

In this section, it is explored the state-of-art and the related works in each field relevant to my research.

Concerning weather prediction, traditional methods such as numerical weather prediction models [26], supplemented by statistical and machine learning models [39, 36, 42, 43, 35], achieve high accuracy. However, understanding models like neural networks remains challenging. Explainable AI (XAI) techniques are being developed to address this issue. For instance, Labe et al. [28] used XAI with ANNs to explain rising summer temperatures in the USA. The idea of my research is, instead, to use ILP since it is able to give explainable results. I am specifically interested in ILASP, which have been already exploited in many study [31] and FastLAS [3, 8]. FastLAS, effective in prioritizing specific rules over general ones [20], has also been integrated with neural networks [9]. Within methods providing explainability, Alviano et al. [2] and Cabalar et al. [4] contributed with some methods such as XASP.

In the legal field, Allen [1] pioneered legal document interpretation using symbolic logic. Then, Kowalski and Sergot [27] classified legal rules, applying logic programming to model British laws. Golshani [23] emphasized argument construction in automated legal reasoning while a more recent work by Sartor et al. [41] utilized Logical English and top-down ASP solvers for legal encoding.

In image recognition, the YOLO network [40] excels in real-time object detection, such as applied in autonomous driving [24]. Of particular interest are instead, the study by Chen et al. [6] and the work by Yang et al. [44] where YOLO is applyed for cell and cancer detection.

Summing up, the usage of ILP in weather prediction provides significant advantages over other methods. Specifically, ILP allows learning concepts that can be interpretable and adaptable to changing conditions. Moreover, unlike related works in the legal and image recognition domains, which primarily focus on statistical and deep learning approaches, our application of ILP introduces a level of transparency and interpretability that these other techniques lack. For example, in the legal domain, ILP can model complex rules and account for vagueness, while in image recognition, it complements neural networks by providing explainable reasons behind classification results, a feature rarely seen in standard approaches.

4 Research goals

The main goal of my research is to integrate deep learning with logic programming to create explainable AI systems. This involves leveraging logic programming, particularly Inductive Logic Programming (ILP), to model complex systems and achieve results that exhibit both high accuracy and explainability. When ILP is not able to reach high levels of accuracy, I want to employ deep learning system for the prediction tasks, due to its superior performance, followed by the application of logic programming to develop systems that can explain their decisions.

In weather prediction, we are employing a hybrid learning approach, where sub-symbolic models (such as RNNs) will handle pattern recognition and time-series forecasting, while ILP will apply symbolic reasoning to provide post-hoc explanations for the RNN predictions. This approach falls under the *'Post-hoc Explainability'* category of the taxonomy [5]. The objective is to develop a system that can analyze meteorological data and generate logic rules that explain them and then use them to explain the outcome of the RNN model (which has an higher accuracy). In the future, we do not exclude the possibility to integrate a third system that will use Natural Language Processing to generate text-based explanations of weather forecasts, similar to those provided by professional meteorologists.

In the legal field, I am employing ILASP to model and reason about complex legal rules and cases. This project aims to develop systems that offer transparent and understandable logic behind automated decisions, thus enhancing trust and transparency in legal AI applications. The final model should assist legal professionals by basing its outcomes on numerous criminal records and providing explanations for the reasoning behind the final judgement. In this field, deep learning techniques would likely be used not to predict outcomes (since laws can be easily translated into logic rules) but to automatically encode articles of the legal code, given the vast amount of data.

For image recognition of bull's spermatozoa, my current focus is on using the YOLO (You Only Look Once) network for object detection. Future work will involve integrating ILP to improve the explainability of these systems, ensuring that the AI not only identifies objects accurately but also provides understandable reasons for its classifications.

5 Current status of research

My ongoing research and current work in each area of interest will be outlined in this section, along with citations of our publications that have shown promising results.

In the domain of weather prediction, we have successfully developed a neural network capable of predicting the number of lightning strikes in the Friuli Venezia Giulia (IT) region with good accuracy. Initially, we used CNNs for this task and have now begun exploring recurrent neural networks to further enhance predictive performance. Our first results [18] indicate that the neural network performs well, though it requires some modifications to improve its robustness. Additionally, we have created a model using FastLAS that allows us to explain the predicted rainfall levels for the upcoming hours. Data was collected for two sets of three months each. Key atmospheric variables recorded include rain, temperature, humidity, wind speed, and pressure. The training process uses a 10-fold cross-validation approach, with each fold containing four days of training data and the remaining days used for validation. The hypothesis space is designed to include rules referencing past conditions to predict future states. This approach encourages the use of predicates indicating changes over time, while limiting the complexity of rules to maintain efficiency. In facts, we prioritizes rules that incorporate past information, using a scoring function to penalize predicates that do not reference multiple timestamps. The results of the experiments using FastLAS have been compared to other models such as SVM, RandomForest, and Decision Tree: although the accuracy of this system is not yet optimal, our initial findings indicate that FastLAS can be effectively used for this purpose since it can often reach the same accuracy as the other systems. The application of xASP is shown in Fig. 1.

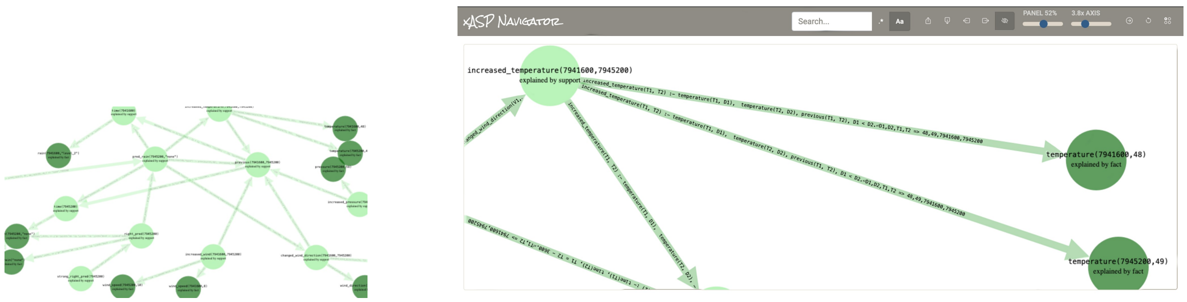


Figure 1: Full explanation of the answer set and zoomed-in view of a predicate explanation by xASP

In the legal field, we have made significant progress by modeling four articles of the Italian Constitution and more or less a hundred precedent cases. This progress ensures that our system can adapt and improve its reasoning capabilities letting ILASP learn from historical legal cases. While the application of ILASP has not been wide so far, it provides a solid basis for further exploration for explainability [10]. Additionally, we are addressing the inherent challenge of vagueness. This involves developing methods to handle ambiguous and context-dependent terms within legal texts, ensuring that our models can reason accurately despite these complexities. To address this issue we construct a model that enables the user to get all the different combination a vague concept can led to, exploiting choices rules. The evaluation of this model showed that it was successfully able to capture the legal distinctions and provide accurate classifications of cases. The results were validated against a set of real-world legal cases, demonstrating the model's ability to interpret complex legal scenarios and make decisions aligned with legal reasoning. Moreover, some incoherence and discrepancies in previous cases were discovered thanks to the system. The ASP model's output included detailed explanations for each classification, highlighting the rules and criteria applied. We presented our preliminary results in [16, 19, 12].

In both weather and legal domain, we are integrating explainability systems, specifically the one developed by Alviano et al. [2], to ensure that our models provide transparent and understandable outputs. Indeed xASP is able to generate directed acyclic graphs which are particularly useful for representing dependencies and causal relationships within a logic program. This could be used in the future as a base to generate text that explain the outcomes.

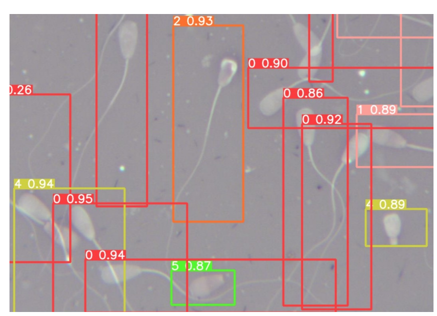


Figure 2: YOLO result on bull's spermatozoa image

The image recognition in our research is still at the beginning stages, but you can see an example of detection our system is currently able to perform in Fig.2. We have developed a basic neural network (using YOLO) that, despite its simplicity, is yielding good results (the accuracy of the current model reached 68%). Our next step is to make these systems more reliable by integrating ILASP, as we already explored how it works [17]. This integration aims to enhance the explainability and robustness of our image recognition systems.

6 Conclusions

This research project investigates the integration of deep learning and logic programming to develop AI systems that combine accuracy with explainability. By leveraging the strengths of neural networks for predictive tasks and ILP tools like FastLAS and ILASP for logical reasoning, the study aims to create interpretable models across various domains.

In weather prediction, the hybrid approach of combining FastLAS with recurrent neural networks aims to deliver accurate forecasts and clear, human-readable explanations, thus enhancing trust in the predictions. In the legal domain, ILASP is employed to model and reason about complex legal rules, providing transparent logic behind decisions and aiding legal professionals by offering understandable explanations for judgments. Finally, for image recognition, integrating YOLO networks with ILP improves AI interpretability, ensuring accurate object detection with clear reasoning.

Additionally, we are working towards making FastLAS compatible with GPU acceleration [38, 15, 14, 13] to enhance computational efficiency and scalability in AI applications. The pursuing of the research will hopefully bring advance in the field of AI and explainable AI.

Acknowledgments. This research is partially supported by Interdepartment Project on AI (Strategic Plan UniUD–2022-25), by NextGenerationEU-PNRR project *MaPSART*-"Future Artificial Intelligence Research", and by INdAM-GNCS 2024 project LCXAI: Logica Computazionale per eXplainable Artificial Intelligence.

References

[1] Layman 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] Mario Alviano, Ly Ly T. Trieu, Tran Cao Son & Marcello Balduccini (2023): *Advancements in xASP, an XAI System for Answer Set Programming*. In: Proceedings of the 38th Italian Conference on Computational Logic, Udine, Italy, June 21-23, 2023, CEUR Workshop Proceedings 3428, CEUR-WS.org. Available at `https://ceur-ws.org/Vol-3428/paper2.pdf`.

[3] Kexin Gu Baugh, Nuri Cingillioglu & Alessandra Russo (2023): *Neuro-symbolic Rule Learning in Real-world Classification Tasks*. In: Proc. of AAAI-MAKE 2023, CEUR Workshop Proceedings 3433, CEUR-WS.org, doi:10.48550/arXiv.2303.16674.

[4] Pedro Cabalar, Jorge Fandinno & Michael Fink (2014): *Causal graph justifications of logic programs*. Theory and Practice of Logic Programming 14(4-5), pp. 603–618, doi:10.48550/arXiv.1409.7281.

[5] Roberta Calegari, Giovanni Ciatto & Andrea Omicini (2020): *On the integration of symbolic and sub-symbolic techniques for XAI: A survey*. Intelligenza Artificiale 14(1), pp. 7–32, doi:10.3233/IA-190036.

[6] Xi Chen, Haoyue Zheng, Haodong Tang & Fan 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.

[7] Andrew Cropper & Sebastijan Dumančić (2022): *Inductive logic programming at 30: a new introduction*. Journal of Artificial Intelligence Research 74, pp. 765–850, doi:10.48550/arXiv.2008.07912.

[8] Daniel Cunnington, Flaviu Cipcigan, Rodrigo Neumann Barros Ferreira & Jonathan Booth (2023): *Symbolic Learning for Material Discovery*. arXiv preprint arXiv:2312.11487, doi:10.48550/arXiv.2312.11487.

[9] Daniel Cunnington, Mark Law, Jorge Lobo & Alessandra Russo (2023): *FFNSL: Feed-forward neural-symbolic learner*. Machine Learning 112(2), pp. 515–569, doi:10.48550/arXiv.2106.13103.

[10] Fabio A D'Asaro, Matteo Spezialetti, Luca Raggioli & Silvia Rossi (2020): *Towards an inductive logic programming approach for explaining black-box preference learning systems*. In: *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, pp. 855–859, doi:10.24963/kr.2020/88.

[11] Shi Dong, Ping Wang & Khushnood Abbas (2021): *A survey on deep learning and its applications*. *Computer Science Review* 40, p. 100379, doi:10.1016/j.cosrev.2021.100379.

[12] Agostino Dovier, Talissa Dreossi & Andrea 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`.

[13] Agostino Dovier, Andrea Formisano, Gopal Gupta, Manuel V. Hermenegildo, Enrico Pontelli & Ricardo Rocha (2022): *Parallel Logic Programming: A Sequel*. *Theory Pract. Log. Program.* 22(6), pp. 905–973, doi:10.1017/S1471068422000059.

[14] Agostino Dovier, Andrea Formisano & Enrico Pontelli (2018): *Parallel Answer Set Programming*. In: *Handbook of Parallel Constraint Reasoning*, Springer, pp. 237–282, doi:10.1007/978-3-319-63516-3_7.

[15] Agostino Dovier, Andrea Formisano, Enrico Pontelli & Flavio Vella (2016): *A GPU Implementation of the ASP Computation*. In: *Practical Aspects of Declarative Languages - 18th International Symposium, PADL 2016, St. Petersburg, FL, USA, January 18-19, 2016. Proceedings*, *Lecture Notes in Computer Science* 9585, Springer, pp. 30–47, doi:10.1007/978-3-319-28228-2_3.

[16] Manuele Dozzi, Talissa Dreossi, Federico Costantini, Agostino Dovier & Andrea Formisano (2023): *Semi-automatic knowledge representation and reasoning on vagueness crime concepts*. Workshop at Jurix 2023.

[17] Talissa Dreossi (2023): *Exploring ILASP Through Logic Puzzles Modelling*. In: *CEUR Workshop Proceedings*, 3428, CEUR-WS. Available at `https://ceur-ws.org/Vol-3428/paper14.pdf`.

[18] Talissa Dreossi, Agostino Dovier, Andrea Formisano, Mark Law, Agostino Manzato, Alessandra Russo & Matthew Tait (2024): *Towards Explainable Weather Forecasting Through FastLAS*. In: *Logic Programming and Nonmonotonic Reasoning - 17th International Conference, LPNMR 2024, Dallas, Texas, USA, October 11-14, 2024, Proceedings*, Springer.

[19] Talissa Dreossi, Manuele Dozzi, Luca Baron, Agostino Dovier, Andrea Formisano & Federico Costantini (2024): *Semi-automatic knowledge representation and reasoning on vague crime concepts*. In: *Book of Abstracts, ESCIM 2024, Krakow*, p. 31. Available at `http://escim2024.uca.es/wp-content/uploads/Book_of_abstracts_ESCIM_2024.pdf#page=45`.

[20] Arthur Drozdov, Mark Law, Jorge Lobo, Alessandra Russo & Mercion Wilathgamuwage Don (2021): *Online Symbolic Learning of Policies for Explainable Security*. In: *2021 Third IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications (TPS-ISA)*, pp. 269–278, doi:10.1109/TPSISA52974.2021.00030.

[21] 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), p. 114–203, doi:10.1017/S1471068418000534.

[22] Michael Gelfond & Vladimir Lifschitz (1988): *The stable model semantics for logic programming*. In: *ICLP/SLP*, 88, Cambridge, MA, pp. 1070–1080, doi:10.2307/2275201.

[23] Forouzan Golshani (1991): *Automated construction of legal arguments*. *Int. J. Intell. Syst.* 6(6), pp. 673–685, doi:10.1002/INT.4550060605.

[24] Lei Huang & Wenzhun Huang (2022): *RD-YOLO: An effective and efficient object detector for roadside perception system*. *Sensors* 22(21), p. 8097, doi:10.3390/s22218097.

[25] Peiyuan Jiang, Daji Ergu, Fangyao Liu, Ying Cai & Bo Ma (2022): *A Review of Yolo algorithm developments*. *Procedia computer science* 199, pp. 1066–1073, doi:10.1016/j.procs.2022.01.135.

[26] Eugenia Kalnay (2002): *Atmospheric Modeling, Data Assimilation and Predictability*. Cambridge University Press, doi:10.1017/CBO9780511802270.

[27] Robert A. Kowalski & Marek J. Sergot (1985): *Computer Representation of the Law*. In: Proceedings of the 9th IJCAI. Los Angeles, CA, USA, August 1985, Morgan Kaufmann, pp. 1269–1270. Available at `http://ijcai.org/Proceedings/85-2/Papers/114.pdf`.

[28] Zachary M Labe, Nathaniel Johnson & Thomas L Delworth (2023): *Changes in United States summer temperatures revealed by explainable neural networks*. Authorea Preprints.

[29] Mark Law (2023): *Conflict-driven inductive logic programming*. Theory and Practice of Logic Programming 23(2), pp. 387–414, doi:10.48550/arXiv.2101.00058.

[30] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda & Jorge Lobo (2020): *Fastlas: Scalable inductive logic programming incorporating domain-specific optimisation criteria*. In: Proc. of the AAAI conference on artificial intelligence, 34, pp. 2877–2885, doi:10.1609/aaai.v34i03.5678.

[31] Mark Law, Alessandra Russo & Krysia Broda (2014): *Inductive learning of answer set programs*. In: Logics in Artificial Intelligence, JELIA 2014, Springer, pp. 311–325, doi:10.1007/978-3-319-11558-0_22.

[32] Mark Law, Alessandra Russo & Krysia Broda (2018): *Inductive learning of answer set programs from noisy examples*. arXiv preprint arXiv:1808.08441, doi:10.48550/arXiv.1808.08441.

[33] Mark Law, Alessandra Russo & Krysia Broda (2020): *The ILASP system for inductive learning of answer set programs*. arXiv preprint arXiv:2005.00904, doi:10.48550/arXiv.2005.00904.

[34] Yann LeCun, Yoshua Bengio & Geoffrey Hinton (2015): *Deep learning*. nature 521(7553), pp. 436–444, doi:10.1038/nature14539.

[35] Lizao Li, Robert Carver, Ignacio Lopez-Gomez, Fei Sha & John Anderson (2024): *Generative emulation of weather forecast ensembles with diffusion models*. Science Advances 10(13), doi:10.1126/sciadv.adk4489.

[36] Amy McGovern (2017): *Using artificial intelligence to improve real-time decision-making for high-impact weather*. Bulletin of the American Meteorological Society 98(10), pp. 2073–2090, doi:10.1175/BAMS-D-16-0123.1.

[37] Stephen H. Muggleton (1991): *Inductive Logic Programming*. New Gener. Comput. 8(4), pp. 295–318, doi:10.1007/BF03037089.

[38] Alessandro Dal Palù, Agostino Dovier, Andrea Formisano & Enrico Pontelli (2015): *CUD@SAT: SAT solving on GPUs*. J. Exp. Theor. Artif. Intell. 27(3), pp. 293–316, doi:10.1080/0952813X.2014.954274.

[39] Stephan Rasp, Michael S Pritchard & Pierre Gentine (2018): *Deep learning to represent subgrid processes in climate models*. Proceedings of the national academy of sciences 115(39), pp. 9684–9689, doi:10.1073/pnas.1810286115.

[40] Joseph Redmon, Santosh Divvala, Ross Girshick & Ali Farhadi (2016): *You only look once: Unified, real-time object detection*. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 779–788, doi:10.1109/CVPR.2016.91.

[41] Galileo Sartor, Jacinto A. Dávila, Marco Billi, Giuseppe Contissa, Giuseppe Pisano & Robert A. Kowalski (2022): *Integration of Logical English and s(CASP)*. In: Proceedings of the ICLP 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`.

[42] Jonathan A Weyn, Dale R Durran & Rich Caruana (2019): *Can machines learn to predict weather? Using deep learning to predict gridded 500-hPa geopotential height from historical weather data*. J. of Advances in Modeling Earth Systems 11(8), pp. 2680–2693, doi:10.1029/2019MS001705.

[43] Jonathan A Weyn, Dale R Durran & Rich Caruana (2020): *Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere*. J. of Advances in Modeling Earth Systems 12(9), doi:10.1029/2020MS002109.

[44] Yunfeng Yang & Jiaqi Wang (2024): *Research on breast cancer pathological image classification method based on wavelet transform and YOLOv8*. Journal of X-Ray Science and Technology, pp. 1–11, doi:10.3233/XST-230296.

Computational methods for Dynamic Answer Set Programming

Susana Hahn

University of Potsdam, Germany
hahnmartinlu@uni-potsdam.de

1 Introduction

In our daily lives, we commonly encounter problems that require reasoning with time. For instance, planning our day, determining our route to work, or scheduling our tasks. We refer to these problems as 'dynamic' because they involve movement and change over time, which sometimes includes metric information to express deadlines and durations. For example, getting to the office within the next hour while ensuring that you have had breakfast beforehand. In industrial settings, the complexity of these problems increases significantly. We see this complexity in scenarios such as train scheduling, production sequencing, and many other operations. Therefore, modeling these large-scale problems requires addressing both dynamic aspects and complex combinatorial optimizations, which is a significant challenge.

Semantic formalisms for expressing dynamic knowledge have been around for many years. Dynamic logics provide the means to describe ordered events, making them powerful tools for domains that need to capture actions and changes. These formalisms are typically approached from a theoretical perspective, and the systems built around them tend to be single-purpose, lacking the flexibility to fully model complex problems. This creates a need for systems that offer comprehensive modeling capabilities for dynamic domains, efficient solving techniques, and tools for industrial integration. Answer Set Programming (ASP) is a prime candidate for solving knowledge-intensive search and optimization problems. This declarative approach offers a rich modeling language and effective solvers. However, ASP is primarily suited for static knowledge and lacks built-in solutions for managing dynamic knowledge.

The overall goal of this research project is to extend ASP into a general-purpose technology for dynamic domains. The first step is to develop the logical foundations for enhancing ASP's base logic with concepts from dynamic, temporal, and metric logic. Significant progress in this area has already been made by previous efforts of our research group, providing a solid foundation for further development. We need to identify fragments of these languages that offer the necessary modeling power for our target dynamic problems while maintaining properties that allow for formalization and translation using various approaches. These approaches include using different structures, such as automata and other transition systems, and compiling durations into other formalisms, such as linear constraints. Implementing these approaches will leverage existing technology in the ASP system *clingo* and its surrounding tools. This project will employ advanced programming techniques in ASP to create effective systems for modeling complex dynamic problems. Additionally, we aim to add interactive capabilities to these systems to benefit both modelers and end users. We anticipate that incorporating these features into ASP will enhance users' ability to model dynamic problems and perform various reasoning tasks.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 324–331, doi:10.4204/EPTCS.416.32

2 Background

Dynamic logics

One of the most commonly used temporal logics [21] is Linear-time Temporal Logic (LTL) [35]. LTL provides modal operators to express temporal properties such as $\circ$ (next), $\square$ (always), and $\lozenge$ (eventually). LTL can also be defined in terms of Dynamic Logic (DL) [27], which allows for writing regular expressions over (infinite) traces and mixing declarative with procedural specifications. These types of specifications are also targeted by action languages such as GOLOG [40], which is based on situation calculus. Another interesting approach is Metric Temporal Logic (MTL) [38], which allows measuring time differences between events. This measurement is done by assigning a time value to states. Metric Logic can be used in different applications such as scheduling [41], routing [37], and more [31]. Originally, these temporal formalisms were investigated for infinite traces. However, in the past decade, the case of finite traces $\cdot_f$ has gained interest, as it aligns with a large range of AI applications and constitutes a computationally more feasible variant. The introduction of LTL_f and Linear Dynamic Logic over finite traces (LDL_f) [19] served as a stepping stone to define the syntax and classical semantics under this restriction.

There are several tasks addressed by these formalisms and other action theories. The most commonly known are satisfiability checking, model checking, and synthesis [47, 48, 20, 49]. Furthermore, other more elaborate tasks closer to real-world scenarios include reactive control [8], diagnostics [33], planning [4, 25, 5], and verification.

Many of these tasks are solved by translating complex constructs into simpler ones, for instance, by reducing MTL into LTL [31]. Another very common strategy for addressing these problems is mapping them into automata. This automata-theoretic approach involves constructing an automaton that accepts exactly the models of a dynamic formula. This relationship has been extensively researched in areas such as satisfiability checking, model checking, planning [4, 25, 5], and synthesis [47, 48, 20, 49]. Non-deterministic finite automata (NFA) [30] and Deterministic Finite Automata (DFA) have been used for finite traces, though they are of exponential size relative to the input formula. To tackle this issue, [19] proposed a translation from LTL_f and LDL_f to a more elaborate but succinct automaton: Alternating Automaton on Finite Words (AFA) [18, 48, 19]. These automata, an adaptation of Alternating Büchi Automata to finite traces, extend NFAs with universal transitions. This translation, however, led to circular definitions for some dynamic formulas and did not include past operators. These issues were addressed in [45], where the authors introduced Automata Linear Dynamic Logic over finite traces (ALDL_f) and presented a translation into even more sophisticated automata: Two-Way Alternating Finite Automata (2AFA) [39, 36]. In addition to alternation, this type of automaton allows multiple head movements: stationary, left, and right. More evolved translations from metric logic into automata have also been developed, such as translating MTL into Timed Automata [42].

Answer Set Programming

Answer Set Programming (ASP) [10] is a well-established approach to declarative problem-solving where problems are encoded as logic programs. The combination of its rich modeling language and highly effective solving engines makes ASP a very attractive choice. ASP semantics can be formalized using equilibrium models [44] of the logic of here-and-there (HT) [29]. This logic has also been extended to here-and-there with constraints (HT_c) [17], which introduces difference constraints, a simplified form of linear constraints, into HT.

The ASP system *clingo* [24] is known for its high-performing engines. The system provides various tools for extending the language and customizing the solving process. *clingo*'s theory language capabilities allow for defining custom syntactic expressions. Additionally, *clingo* offers two methods for capturing new functionalities [34]: meta-programming, which uses a reification feature enabling the expression of new functionalities using ASP, and a sophisticated Python API for manipulating and customizing the system's internal workflow. This customization includes techniques such as multi-shot solving, which allows precise control of the solving process by modularizing the problem. These features have led to the creation of several hybrid ASP systems. In particular, *clingcon* [7] and *clingo*[DL] [32] extend the language of *clingo* with linear constraints using the semantics for HT_c.

Temporal Logic Programming

In the 1980s, Temporal Logic Programming (TLP) emerged [22, 23, 1, 43]. TLP was revised after the appearance of ASP, resulting in what we know as Temporal ASP. The idea is to extend the equilibrium models of HT, to deal with dynamic scenarios. Research began with infinite traces, giving rise to (Linear) Temporal Here-and-There (THT) [3] and (Linear) Dynamic Logic of Here-and-There (DHT) [9], along with their non-monotonic counterparts for temporal stable models, namely Temporal Equilibrium Logic (TEL) [3] and Dynamic Equilibrium Logic (DEL) [14]. The strategy behind these temporal formalisms is to capture time as sequences of HT-interpretations, resulting in an expressive non-monotonic modal logic. This approach allowed the definition of Temporal Logic Programs found in [2].

The temporal operators and semantics of the finite version TEL_f were introduced into the ASP system *clingo* enriching its modeling power and yielding the first temporal ASP solver *telingo* [16]. This system uses the *clingo* capabilities for theory extensions as well as multi-shot solving in an incremental manner. Subsequent work incorporated dynamic operators from DEL_f [14, 13] by unfolding their definitions into TEL_f relying on the introduction of auxiliary atoms (in a Tseitin-style [46]). This technique, however, is dependent on a fixed trace length, and the type of translation makes the final logic program hard to interpret. In [15], TEL was further extended with metric temporal operators constrained by intervals over natural numbers, resulting in Metric Equilibrium Logic (MEL).

3 Contributions and future work

Automata techniques

To this moment, I have pursued different translations of dynamic and temporal logic with finite traces into automata, and implemented them using ASP. In the current status of the project, I have not yet explored the non-monotonic side of temporal reasoning with automata. Even though the semantics I have used so far for the temporal formalisms have been monotonic, I was able to incorporate them in ASP by restricting the dynamic formulas to integrity constraints where their behavior is classical. With this in mind, at the moment, one can only use these formulas to filter plans via a translation into automata, which is one of our primary goals.

The first approach, found in [11], proposes an adaptation of the translation of LDL_f to AFA from [19], which is incorporated into ASP using meta programming in *clingo*. This implementation is solely based on ASP, relying on the theory extension to define the language for LDL_f formulas, and the reified output of *clingo*. This reification corresponds to a linearized representation of the dynamic formula as facts based on the grammar defined for the theory. Then, using an ASP encoding, these facts are translated into a declarative representation of the corresponding alternating automaton. In the full version of this work [12], we explore other existing tools for computing an automaton from a dynamic formula. In order to employ them, we developed two different translations from LDL_f to Monadic Second Order Logic (MSO). For the implementation, we parsed the formula with *clingo*'s API and called the state-of-the-art system MONA [28] to obtain a DFA, which is then transformed into facts. By having a unified declarative representation to capture the different automata (AFA and DFA), we were able to craft a single encoding for checking the acceptance of the automata.

Following (soon to be published) work presents a novel translation from LDL_f into 2AFA. Leveraging the transitions without head movement provided by these automata, we were able to remove the recursive nature of our old translation, thus eliminating the circular issue carried from [19]. Furthermore, the left head movement allows us to readily refer to time points in the past. These new target automata, nonetheless, represented a formalization challenge since there is a lack of literature available in contrast to simpler automata. This translation was implemented as part of the `adlingo`[1] system. Just like the previous work, the implementation was done using meta programming and theory extensions. Additionally, it integrates the use of *clingraph* [26] to visualize the automata and its runs using an ASP encoding.

Linear constraints to encode durations

My latest work has focused on constructing the foundations of Metric Logic Programs (MLP). With these programs, we plan to abstract the basic modalities and forms needed to model metric problems in ASP. The semantics of these programs are those of MEL, where, by restricting the syntax, we aim to simplify the computation. The first approach for this work has been submitted to ICLP24. As in the automata approach, we are restricting the research to the finite setting for a given (fixed) horizon. This work defines the basic syntax for a MLP in which all rules are universal, meaning that they must hold in every step. This contrasts with the approach used for temporal logic programs in [2], where the rules were separated into initial, dynamic, and final, which facilitates the implementation using incremental solving. For our approach, the removal of this division simplifies the use of meta-programming techniques to define the translation, as well as the overall semantics. The use of meta-programming allows us to define the translations in ASP, making the implementation transparent and modular. A big advantage of this approach is the clear mapping between the translation and the implementation in ASP. As a consequence, it simplifies the proves of correctness and completeness of the translation.

In this first exploration, we restricted the rules of metric logic programs to only use the metric next modality. For instance, with the rule $\circ_{[20..40)} school \leftarrow drive$, we express that *"If I start driving, I must be at the school in the next step, which should happen in 20 to 40 minutes"*. We suspect that the core of our target dynamic problems can be modeled with this restricted fragment. In a nutshell, the first part of this translation represents the state changes and follows the same semantics as in TEL_f. The second part accounts for the timed aspect of metric logic. For this part, we explore two approaches: one where the translation is done to HT, and a second one where the target logic is HT_c. As a result, we were able to see what we expected: a succinct and performant translation of time into linear constraints. We also observed that our restricted language did allow us to model the transitions and time restrictions, but was not enough to represent the goal conditions of the problems. These conditions usually require more complex metric operators to talk about states that are further away in time, for instance, *"At some point in the next 2 hours, I will be back home"*.

[1] `https://github.com/potassco/adlingo`

3.1 Future work

The quest to conceptualize metric logic programming is far from over. In view of the results from the last project, we have started to craft a translation that handles more metric operators. The translation is planned to follow a Tseitin-style translation like the one in [2]. We want to further investigate HT_c for encoding time, and examine the integration of non-monotonic reasoning and optimization in the timed aspect of MLP. Additionally, we plan to investigate how far ASP can address reactive-dynamic tasks where the user and environment play a role by interacting with the system.

References

[1] M. Abadi & Z. Manna (1989): *Temporal Logic Programming*. Journal of Symbolic Computation 8, pp. 277–295, doi:10.1016/S0747-7171(89)80070-7.

[2] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez, T. Schaub, A. Schuhmann & C. Vidal (2023): *Linear-Time Temporal Answer Set Programming*. Theory and Practice of Logic Programming 23(1), pp. 2–56, doi:10.1017/S1471068421000557.

[3] F. Aguado, P. Cabalar, M. Diéguez, G. Pérez & C. 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.

[4] J. Baier, C. Fritz, M. Bienvenu & S. McIlraith (2008): *Beyond Classical Planning: Procedural Control Knowledge and Preferences in State-of-the-Art Planners*. In D. Fox & C. Gomes, editors: Proceedings of the Twenty-third National Conference on Artificial Intelligence (AAAI'08), AAAI Press, pp. 1509–1512. Available at `https://auld.aaai.org/Library/AAAI/2008/aaai08-251.php`.

[5] J. Baier & S. McIlraith (2006): *Planning with First-Order Temporally Extended Goals using Heuristic Search*. In Y. Gil & R. Mooney, editors: Proceedings of the Twenty-first National Conference on Artificial Intelligence (AAAI'06), AAAI Press, pp. 788–795. Available at `https://www.aaai.org/Library/AAAI/2006/aaai06-125.php`.

[6] M. Balduccini, Y. Lierler & S. Woltran, editors (2019): *Proceedings of the Fifteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'19)*. Lecture Notes in Artificial Intelligence 11481, Springer-Verlag, doi:10.1007/978-3-030-20528-7.

[7] M. Banbara, B. Kaufmann, M. Ostrowski & T. Schaub (2017): *Clingcon: The Next Generation*. Theory and Practice of Logic Programming 17(4), pp. 408–461, doi:10.1017/S1471068417000138.

[8] C. Baral, S. Tran Cao & L. Tuan (2002): *A transition function based characterization of actions with delayed and continuous effects*. In: KR, Citeseer, pp. 291–302. Available at `https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d4ae7ddfcdc012e519b473d03dd3c7caffaa09e1`.

[9] A. Bosser, P. Cabalar, M. Diéguez & T. Schaub (2018): *Introducing Temporal Stable Models for Linear Dynamic Logic*. In M. Thielscher, F. Toni & F. Wolter, editors: Proceedings of the Sixteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'18), AAAI Press, pp. 12–21. Available at `https://aaai.org/ocs/index.php/KR/KR18/paper/view/18047`.

[10] 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.

[11] P. Cabalar, M. Diéguez, S. Hahn & T. Schaub (2021): *Automata for Dynamic Answer Set Solving: Preliminary Report*. In: Proceedings of the Fourteenth Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'21). Available at `https://ceur-ws.org/Vol-2970/aspocpinvited1.pdf`.

[12] P. Cabalar, M. Dieguez, S. Hahn & T. Schaub (2021): *Automata for dynamic answer set solving: Preliminary report*. CoRR abs/2109.01782, doi:10.48550/arXiv.2109.01782 .

[13] P. Cabalar, M. Diéguez, F. Laferriere & T. Schaub (2020): *Implementing Dynamic Answer Set Programming over finite traces*. In G. De Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín & J. Lang,

editors: *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*, IOS Press, pp. 656–663, doi:10.3233/FAIA200151.

[14] P. Cabalar, M. Diéguez & T. Schaub (2019): *Towards Dynamic Answer Set Programming over finite traces*. In Balduccini et al. [6], pp. 148–162, doi:10.1007/978-3-030-20528-7_12.

[15] P. Cabalar, M. Diéguez, T. Schaub & A. Schuhmann (2022): *Metric Temporal Answer Set Programming over Timed Traces*. 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. 117–130, doi:10.1007/978-3-031-15707-3_10.

[16] P. Cabalar, R. Kaminski, P. Morkisch & T. Schaub (2019): *telingo = ASP + Time*. In Balduccini et al. [6], pp. 256–269, doi:10.1007/978-3-030-20528-7_19.

[17] P. Cabalar, R. Kaminski, M. Ostrowski & T. Schaub (2016): *An ASP Semantics for Default Reasoning with Constraints*. In R. Kambhampati, editor: *Proceedings of the Twenty-fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*, IJCAI/AAAI Press, pp. 1015–1021, doi:10.5555/3060621.3060762.

[18] A. Chandra, D. Kozen & L. Stockmeyer (1981): *Alternation*. *Journal of the ACM* 28(1), pp. 114–133, doi:10.1145/322234.322243.

[19] G. De Giacomo & M. Vardi (2013): *Linear Temporal Logic and Linear Dynamic Logic on Finite Traces*. In F. Rossi, editor: *Proceedings of the Twenty-third International Joint Conference on Artificial Intelligence (IJCAI'13)*, IJCAI/AAAI Press, pp. 854–860. Available at `https://www.ijcai.org/Abstract/13/132`.

[20] G. De Giacomo & M. Vardi (2015): *Synthesis for LTL and LDL on Finite Traces*. In Q. Yang & M. Wooldridge, editors: *Proceedings of the Twenty-fourth International Joint Conference on Artificial Intelligence (IJCAI'15)*, AAAI Press, pp. 1558–1564. Available at `https://ijcai.org/Abstract/15/223`.

[21] S. Demri, V. Goranko & M. Lange (2016): *Temporal Logics in Computer Science: Finite-State Systems*. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, doi:10.1017/CBO9781139236119.

[22] M. Fujita, S. Kono, H. Tanaka & T. Moto-Oka (1986): *Tokio: Logic Programming Language Based on Temporal Logic and its Compilation to Prolog*. In E. Shapiro, editor: *Proceedings of the Third International Conference on Logic Programming (ICLP'86)*, *Lecture Notes in Computer Science* 225, Springer, pp. 695–709, doi:10.1007/3-540-16492-8_119.

[23] D. Gabbay (1987): *Modal and Temporal Logic Programming*. In A. Galton, editor: *Temporal Logics and their Applications*, chapter 6, Academic Press, pp. 197–237.

[24] M. Gebser, R. Kaminski, B. Kaufmann & T. Schaub (2019): *Multi-shot ASP solving with clingo*. *Theory and Practice of Logic Programming* 19(1), pp. 27–82, doi:10.1017/S1471068418000054.

[25] G. De Giacomo & S. Rubin (2018): *Automata-Theoretic Foundations of FOND Planning for LTLf and LDLf Goals*. In J. Lang, editor: *Proceedings of the Twenty-seventh International Joint Conference on Artificial Intelligence (IJCAI'18)*, ijcai.org, pp. 4729–4735, doi:10.24963/ijcai.2018/657.

[26] S. Hahn, O. Sabuncu, T. Schaub & T. Stolzmann (2024): *Clingraph: A System for ASP-based Visualization*. *Theory and Practice of Logic Programming*, doi:10.1017/S147106842400005X.

[27] D. Harel, J. Tiuryn & D. Kozen (2000): *Dynamic Logic*. MIT Press, doi:10.1145/568438.568456.

[28] J. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, R. Paige, T. Rauhe & A. Sandholm (1995): *Mona: Monadic Second-Order Logic in Practice*. In E. Brinksma, R. Cleaveland, K. Larsen, T. Margaria & B. Steffen, editors: *Proceedings of the First International Workshop on Tools and Algorithms for Construction and Analysis of Systems (TACAS'95)*, *Lecture Notes in Computer Science* 1019, Springer-Verlag, pp. 89–110, doi:10.1007/3-540-60630-0_5.

[29] 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.

[30] J. Hopcroft & J Ullman (1979): *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

[31] U. Hustadt, A. Ozaki & C. Dixon (2020): *Theorem Proving for Pointwise Metric Temporal Logic Over the Naturals via Translations*. Journal of Automated Reasoning 64(8), pp. 1553–1610, doi:10.1007/s10817-020-09541-4.

[32] T. Janhunen, R. Kaminski, M. Ostrowski, T. Schaub, S. Schellhorn & P. Wanko (2017): *Clingo goes Linear Constraints over Reals and Integers*. Theory and Practice of Logic Programming 17(5-6), pp. 872–888, doi:10.1017/S1471068417000242.

[33] S. Jiang & R. Kumar (2004): *Failure diagnosis of discrete-event systems with linear-time temporal logic specifications*. IEEE Transactions on Automatic Control 49(6), pp. 934–945, doi:10.1109/TAC.2004.829616.

[34] R. Kaminski, T. Schaub & P. Wanko (2017): *A Tutorial on Hybrid Answer Set Solving with clingo*. In G. Ianni, D. Lembo, L. Bertossi, W. Faber, B. Glimm, G. Gottlob & S. Staab, editors: Proceedings of the Thirteenth International Summer School of the Reasoning Web, Lecture Notes in Computer Science 10370, Springer-Verlag, pp. 167–203, doi:10.1007/978-3-319-61033-7_6.

[35] J. Kamp (1968): *Tense Logic and the Theory of Linear Order*. Ph.D. thesis, University of California at Los Angeles.

[36] C. Kapoutsis & M. Zakzok (2021): *Alternation in two-way finite automata*. Theoretical Computer Science 870, pp. 75–102, doi:10.1016/j.tcs.2020.12.011.

[37] S. Karaman & E. Frazzoli (2008): *Vehicle routing problem with metric temporal logic specifications*. In: 2008 47th IEEE conference on decision and control, IEEE, pp. 3953–3958, doi:10.1109/CDC.2008.4739366.

[38] R. Koymans (1990): *Specifying Real-Time Properties with Metric Temporal Logic*. Real-Time Systems 2(4), pp. 255–299, doi:10.1007/BF01995674.

[39] R. Ladner, R. Lipton & L Stockmeyer (1984): *Alternating pushdown and stack automata*. SIAM Journal on Computing 13(1), pp. 135–155, doi:10.1137/0213010.

[40] H. Levesque, R. Reiter, Y. Lespérance, F. Lin & R. Scherl (1997): *GOLOG: A Logic Programming Language for Dynamic Domains*. Journal of Logic Programming 31(1-3), pp. 59–83, doi:10.1016/S0743-1066(96)00121-5.

[41] R. Luo, R. Valenzano, Y. Li, C. Beck & S. McIlraith (2016): *Using Metric Temporal Logic to Specify Scheduling Problems*. In C. Baral, J. Delgrande & F. Wolter, editors: Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'16), AAAI Press, pp. 581–584. Available at `https://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12909`.

[42] D. Ničković & N. Piterman (2010): *From MTL to Deterministic Timed Automata*. In K. Chatterjee & T. Henzinger, editors: Proceedings of the Eighth International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'10), Lecture Notes in Computer Science, Springer-Verlag, pp. 152–167, doi:10.1007/978-3-642-15297-9_13.

[43] M. Orgun & W. Wadge (1992): *Theory and Practice of Temporal Logic Programming*. In L. Fariñas del Cerro & M. Penttonen, editors: Intensional Logics for Programming, chapter 2, Oxford University Press, pp. 21–50, doi:10.1093/oso/9780198537755.003.0002.

[44] D. Pearce (2006): *Equilibrium logic*. Annals of Mathematics and Artificial Intelligence 47(1-2), pp. 3–41, doi:10.1007/s10472-006-9028-z.

[45] K. Smith & M. Vardi (2021): *Automata Linear Dynamic Logic on Finite Traces*. arXiv preprint arXiv:2108.12003.

[46] G. Tseitin (1968): *On the complexity of derivation in the propositional calculus*. Zapiski nauchnykh seminarov LOMI 8, pp. 234–259.

[47] M. Vardi (1995): *An Automata-Theoretic Approach to Linear Temporal Logic*. In F. Moller & G. Birtwistle, editors: Logics for Concurrency: Structure versus Automata, Lecture Notes in Computer Science 1043, Springer-Verlag, pp. 238–266, doi:10.1007/3-540-60915-6_6.

[48] M. Vardi (1997): *Alternating Automata: Unifying Truth and Validity Checking for Temporal Logics*. In W. McCune, editor: Proceedings of the Fourteenth International Conference on Automated Deduction

(CADE'97), *Lecture Notes in Computer Science* 1249, Springer-Verlag, pp. 191–206, doi:10.1007/3-540-63104-6_19.

[49] S. Zhu, G. Pu & M. Vardi (2019): *First-Order vs. Second-Order Encodings for LTLf-to-Automata Translation*. In T. Gopal & J. Watada, editors: *Proceedings of the Fifteenth Annual Conference on Theory and Applications of Models of Computation (TAMC'19)*, *Lecture Notes in Computer Science* 11436, Springer-Verlag, pp. 684–705, doi:10.1007/978-3-030-14812-6_43.

Relating Answer Set Programming and Many-sorted Logics for Formal Verification

Zachary Hansen
University of Nebraska Omaha
Omaha, Nebraska
929 S. 70th Plz #25, Omaha NE, 68106
zachhansen@unomaha.edu

1 Introduction

Answer Set Programming (ASP) is an important logic programming paradigm within the field of Knowledge Representation and Reasoning. As a concise, human-readable, declarative language, ASP is an excellent tool for developing trustworthy (especially, artificially intelligent) software systems. However, formally verifying ASP programs offers some unique challenges, such as

1. a lack of modularity (the meanings of rules are difficult to define in isolation from the enclosing program),
2. the ground-and-solve semantics (the meanings of rules are dependent on the input data with which the program is grounded), and
3. limitations of existing tools.

My research agenda has been focused on addressing these three issues with the intention of making ASP verification an accessible, routine task that is regularly performed alongside program development. In this vein, I have investigated alternative semantics for ASP based on translations into the logic of here-and-there and many-sorted first-order logic. These semantics promote a modular understanding of logic programs, bypass grounding, and enable us to use automated theorem provers to automatically verify properties of programs.

2 Background

The stable model semantics of logic programs can be expressed in a variety of ways, each of which offers unique insights and utility [38]. For instance, logic programs can sometimes be viewed as "shorthand" for propositional, first-order, default, or autoepistemic theories. These translational approaches (which typically involve a syntactic transformation from ASP rules into a "formula representation" that uses the syntax of first-order logic) offer some advantages over their fixpoint relatives. In particular, semantics that let us bypass the issue of grounding are very convenient for program verification purposes. They allow us to disregard the specifics of the input data with which a program is paired when assessing the independent meaning of the program.

Clark's Completion and its subsequent extensions [11, 39] provide a useful translational semantics for a broad class of programs satisfying the *tightness* condition[1]. Under these semantics, the logic program

$$p \text{ :- } q. \quad p \text{ :- } r.$$

[1] A tight program has a predicate dependency graph without positive cycles [20, 14].

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 332–344, doi:10.4204/EPTCS.416.33

is understood as the first-order theory

$$p \leftrightarrow q \vee r$$

which reflects the minimality of circumscription and the closely related principle of rational belief (we believe p only if we have to, that is, if and only if q or r holds) found in the autoepistemic intuitions behind answer set programming [27]. However, completion semantics are only applicable to programs of a limited form – first-order logic cannot, for example, correctly capture the transitive closure of a binary relation without non-standard assumptions like the Closed World Assumption. This restriction can potentially be circumvented or at least relaxed through innovation in the areas of local tightness [21], loop formulas [37], tightening [48], and ordered completion [3].

More recently, a surprisingly deep connection between the logic of here-and-there [32] and ASP has yielded interesting results. The logic of here-and-there extends intuitionistic logic with the axiom schema

$$F \vee (F \rightarrow G) \vee \neg G$$

of which the most commonly employed consequence is the weak law of excluded middle:

$$\neg F \vee \neg\neg F. \tag{1}$$

Here-and-there has a number of useful characteristics that make it applicable to the study of ASP [44]. First, it acts as a well-behaved monotonic basis for equilibrium logic, which provides a non-monotonic inference relation that captures and generalizes the semantics of a broad class of ASP programs to full propositional logic (recall that while ASP rules have a very limited syntactic form, propositional formulas can be very complex). Furthermore, intuitionistic logic cannot be strengthened more than here-and-there and still be strictly contained in classical logic, making it the strongest, well-behaved classical logic available. Finally, the extension to ASP programs with variables is naturally covered by quantified formulas interpreted under the semantics of here-and-there.

One of the most fruitful consequences of this connection is the study of *strong equivalence* [40]. The condition of strong equivalence states that two programs, Π_1 and Π_2, are strongly equivalent if $\Pi_1 \cup \Pi$ and $\Pi_2 \cup \Pi$ have the same answer sets for any program Π. This condition is useful because it tells us that our two programs are truly interchangeable: no matter which program they form a subcomponent of, they can be swapped out without affecting the enclosing program's answer sets. It has been shown that the strong equivalence of two programs can be established by deriving the formula representation of each program from the formula representation of the other within the logic of here-and-there; for propositional programs this can be done in exponential time [40]. However, more complex ASP languages require more sophisticated translations into formula representations, as well as extended deductive systems. For the remainder of this paper, we will focus on a theoretical ASP language known as MINI-GRINGO.

The answer set solver CLINGO implements the language ABSTRACT GRINGO [24], whose semantics are defined by a translation (τ) into the infinitary propositional logic developed by Truszczyński [46]. MINI-GRINGO is an expressive fragment of ABSTRACT GRINGO which supports negation, arithmetic, intervals, and basic choice rules [21]. The semantics of the language can be captured by a syntactic transformation (τ^*) into first-order formula representations, which are interpreted under the HTA (here-and-there with arithmetic) deductive system [19]. An equivalent (and arguably simpler) characterization of these semantics are defined by the SM operator [23]. This operator transforms a program's (Π) formula representation ($\tau^*\Pi$) into a second-order sentence ($SM_{\mathbf{p}}[\tau^*(\Pi)]$) in which predicate quantification is used to minimize belief in the list $\mathbf{p}$ of *intensional* predicates (similar to circumscription). When $\mathbf{p}$ contains all the predicates occurring in the logic program, then the models of $SM_{\mathbf{p}}[\tau^*(\Pi)]$ correspond to the answer sets of Π. Bartholomew and Lee extend the concept of the *SM* operator to intensional functions [5]. This is once again rooted in the logic of here-and-there, where interpretations can be viewed as pairs (of "worlds") $\langle H,T\rangle$. Predicate minimization through the *SM* operator is achieved by mandating that $p^H \subseteq p^T$ for every $p \in \mathbf{p}$, similarly, for each intensional function f we require that $f^H \neq f^T$.

The study of the MINI-GRINGO language is motivated in large part by a desire to use automated reasoning tools for proving the correctness of ASP programs. ANTHEM is a software system that converts MINI-GRINGO programs into first-order formulas of two sorts (a supersort which consists of all program terms, and a subsort corresponding to integers) via the τ^* translation [20]. It then uses the first-order theorem prover VAMPIRE [35] to check the equivalence of the completion of the program ($COMP[\tau^*(\Pi)]$) to a set of first-order formulas acting as a specification (S), under a set of assumptions characterizing the intended program inputs (A). That is, it attempts to establish the universal validity of

$$A \rightarrow (COMP[\tau^*(\Pi)] \leftrightarrow S)$$

which, if it succeeds, is proof that the program Π implements the specification. This procedure applies to *io-programs*, that is, ASP programs paired with placeholders as well as input and output predicate symbols. Any predicate symbols that are neither input nor output symbols are private symbols – these auxiliary symbols are not essential for understanding the program's "external" behavior as characterized by the output symbols. For example, if we take $prime/1$ to be an output symbol, the programs

$$\begin{aligned} composite(I*J) &\text{ :- } I = 2..n,\ J = 2..n. \\ prime(I) &\text{ :- } I = 2..n,\ not\ composite(I). \end{aligned}$$

and

$$\begin{aligned} comp(X) &\text{ :- } X = I*J,\ I = 2..n,\ J = 2..n. \\ prime(I) &\text{ :- } I = 2..n,\ not\ comp(I). \end{aligned}$$

clearly have the same external behavior (the extent of the $prime/1$ predicate) under the assumption that the placeholder n is an integer, despite minor differences in how the auxiliary predicates are defined. This procedure is only possible for programs satisfying the restrictions of tightness and absence of private recursion.[2]

3 Goals

My interest in the relationship of ASP to many-sorted first-order logic (and, more broadly, the logic of here-and-there) is motivated by its application to software verification. My long-term research agenda is to help develop an accessible, tool-assisted methodology for formally verifying the correctness of ASP programs. Clearly, such an agenda would extend beyond the duration of a single dissertation. For this reason, I've organized the remainder of this research summary into sections showcasing which pieces of this plan have already been addressed (Current Status and Preliminary Results), which pieces I plan to investigate during the remainder of my doctoral program (Ongoing Directions and Expected Achievements), and which pieces I hope to build on top of my eventual dissertation (Conclusions and Future Directions).

[2] A program is tight if its predicate dependency graph has no cycles consisting of positive edges such as (p,p). A program has private recursion if the subgraph induced by private symbols has no cycles.

4 Current Status and Preliminary Results

Since joining the University of Nebraska Omaha (UNO) in Fall 2020, I have been engaged in several research projects under the umbrella topic of *formal verification of ASP programs*. Within this topic, I have been fortunate to collaborate with researchers at UNO, University of Texas at Austin, and the University of Potsdam. My frequent collaborators include Dr. Yuliya Lierler (my advisor), Dr. Jorge Fandinno, Dr. Vladimir Lifschitz, Dr. Torsten Schaub, and Tobias Stolzmann (another PhD student writing his dissertation on topics related to ANTHEM). Where appropriate, I will distinguish between collaborative work and work I've done independently.

4.1 Results Presented at Previous Doctoral Consortiums

Conditional literal semantics Conditional literals are a useful feature supported by CLINGO. The following rule (from a Graph Coloring encoding) succinctly expresses that a vertex cannot have no colors assigned to it.

```
:- not asg(V, C) : col(C); vtx(V).
```
(2)

In the spirit of Fandinno et al. (2020), Dr. Lierler and I developed a translation from a simple ASP language that is (mostly) a subset of MINI-GRINGO extended with conditional literals to unsorted first-order logic [31]. This provided formal support for the intuition that conditional literals in this language represent nested implications within rule bodies. For example, the previous rule can be understood as the first-order sentence

$$\forall V\big(\,(\forall C(col(C) \to \neg asg(V,C)) \wedge vtx(V)) \to \bot\big). \tag{3}$$

We demonstrated that these semantics capture the behavior of CLINGO, and used them to prove the correctness of a *k-coloring program*.

Aggregate semantics Dr. Lierler, Dr. Fandinno, and I proposed a characterization of aggregate semantics that bypasses the need for grounding [16]. Instead, we apply a many-sorted generalization of the SM operator to a set of many-sorted first-order formulas ($\kappa\Pi$) representing a logic program (Π). Aggregates are defined as functions on sets of tuples, whose members are restricted to those tuples satisfying the list of conditions present in the associated aggregate. We designed a set of second-order (first-order in the presence of finite aggregates) axioms to define the behavior of sets and aggregate function symbols. For a class of standard interpretations satisfying assumptions such as a standard interpretation of addition, models of $SM[\kappa\Pi]$ satisfying these aggregate axioms are in one-to-one correspondence with the stable models of Π. When Π is tight, the second-order characterization ($SM[\kappa\Pi]$) can be replaced by completion ($COMP[\kappa\Pi]$). Thus, for tight programs with finite aggregates, our proposed semantics define, via first-order logic, the behavior of CLINGO aggregates.

Modular proofs of correctness with aggregate constraints A key challenge in verifying logic programs is proving the correctness of groups of rules in isolation from the rest of the program. A "divide-and-conquer" methodology is very natural for verification, but applying it to logic programs requires a careful methodology, such as the one proposed by Cabalar, Fandinno, and Lierler (2020). They divide their example Hamiltonian Cycle program into various independent modules, whose behavior is captured via the SM operator [6]. We extend their example to the Traveling Salesman problem with the addition of an aggregate constraint (4) on the cumulative weight of the selected cycle, and use our many-sorted semantics for aggregates to verify the behavior of this constraint independently of the Hamiltonian Cycle program [15]. Additionally, we prove the correctness of a Graph Coloring encoding containing choice rules with cardinality bounds. This project showcases how the modular proof methodology can be extended with our proposed aggregate semantics to argue the correctness of a broad class of logic programs.

```
:- #sum{ K,X,Y : in(X,Y), cost(K,X,Y) } > J, maxCost(J).            (4)
```

4.2 New Results

Extending MINI-GRINGO with conditional literals The ABSTRACT GRINGO fragment Dr. Lierler and I investigated in 2022 was unsorted, forbade double negations, and lacked support for arithmetic operations and intervals. These are important features supported by the MINI-GRINGO language on which ANTHEM is based. We are working on extending the full MINI-GRINGO language presented in Fandinno, Lifschitz, and Temple (2024) with conditional literals. The translation of conditional literals is largely the same, but proving the correctness of the translation for this extended language is considerably more complicated. However, doing so allows us to check the strong equivalence of MINI-GRINGO programs with conditional literals. Furthermore, this work is useful because it acts as a roadmap for the more challenging task of extending MINI-GRINGO and ANTHEM with aggregates using our many-sorted first-order characterization.

This research has also yielded some interesting insights into the way conditional literals can be used to eliminate auxiliary predicates. Doing so makes modular programming, and constructing arguments of correctness about modular programs, considerably easier. For example, consider a traditional encoding of the Graph Coloring problem without conditional literals:

```
{asg(V, C)} :- vtx(V), col(C).                  (5)
:- asg(V, C1), asg(V, C2), C1 != C2.            (6)
colored(V) :- asg(V, C).                        (7)
:- vtx(V), not colored(V).                      (8)
:- asg(V1, C), asg(V2, C), edge(V1, V2).        (9)
```

We can eliminate the auxiliary predicate $colored/1$ by replacing rules (7- 8) with the conditional literal constraint (2). This example hints at a more general property. Note that the rule (7) expresses that a property (*colored*) holds for a vertex (V) if and only if there exists an element (C) such that V is mapped to C by the $asg/2$ predicate. This is an indirect way of expressing an existential quantification – in conjunction with rule (8), it expresses that every vertex must be mapped to a color. This condition can be more concisely represented via conditional literal (3), which is classically equivalent to

$$\forall V \big(vtx(V) \rightarrow \exists C(col(C) \wedge asg(V,C))\big).$$

Axiomatizing new aggregates The many-sorted first-order logic characterization of aggregates Dr. Lierler, Dr. Fandinno, and I developed captured the behavior of CLINGO's `count` and `sum` aggregates [16]. Furthermore, for programs adhering to the ASP-Core-2 standard [7], this characterization also captures the ASP-Core-2 semantics. I have since extended this characterization with `min`, `max`, and `sum+` aggregates. This required developing first and second-order axiomatizations for the new aggregates and proving that they correctly captured the behavior of CLINGO. These results, alongside significant extensions such as a detailed section integrating our results with Clark's Completion, were recently published in the Journal of Artificial Intelligence Research [17].

Recursive aggregates The aggregate axiomatization projects described thus far have included a restriction on positive recursion through aggregates. While such recursion is a comparatively rare scenario in practice, it is very important to the study of strong equivalence. Since two programs Π_1 and Π_2 must have the same answer sets when combined with any program Π to be strongly equivalent, any discussion of strongly equivalent logic programs must accommodate the case when Π introduces recursion through the aggregates of Π_1 or Π_2. To address this, Dr. Fandinno and I extended our aggregate semantics with intensional function symbols [30]. We treated aggregates as functions on sets of tuples – these so-called "set symbols," which map ground terms to sets of tuples of ground terms, were treated intensionally. We found that our proposed semantics coincides with the semantics of CLINGO, but naturally diverged from the semantics of DLV in the presence of recursive aggregates. I presented these results at ASPOCP 2023.

Program to program verification The original version of ANTHEM required a specification written in first-order logic. However, it has since become clear that many ASP programmers would rather write their specifications in the form of a simple, easy-to-read ASP program. To support this, I developed the AP2P[3] system on top of the original ANTHEM. The theoretical results supporting this procedure were published in TPLP [18]. AP2P allows users to automatically confirm the (external) equivalence of two ASP programs. This is primarily useful in the refactoring process, wherein a simple program may be successively replaced by more complex programs in the interest of improving performance. Our system checks that the essential behavior of the program has not been changed during refactoring.

Anthem 2 The long-term viability of the prototypical ANTHEM system was threatened by technical debt such as a lack of documentation, heavy dependence on deprecated Rust nightly features, and a divergence of the system's behavior from the supporting theory. I have been working in collaboration with Dr. Lifschitz and Tobias Stolzmann on a complete overhaul and re-implementation of this prototype that corrects and extends it in several ways. First, I have restructured and generalized the verification process to enable a symmetric treatment of program-to-program and program-to-specification verification. This positions ANTHEM as a tool to support refactoring ASP code in addition to its original functionality. Second, Tobias Stolzmann and I have corrected the internal representation of many-sorted first-order theories and the partial axiomatizations of standard interpretations. This eases (in particular) the handling of placeholders in io-programs. Third, I have designed and implemented a suite of transformations equivalent in the logic of here-and-there to simplify the formulas being passed to the backend theorem prover. Preliminary experiments show substantial improvements in runtime for certain programs. Fourth, Dr. Lifschitz and I have designed and implemented an improved control language for writing proof outlines, which offers users a considerably more fine-grained control over the formulation of verification tasks. This moves ANTHEM away from a one-shot system towards an interactive proof assistant, which is crucial for verifying non-trivial problems. Finally, I wrote a reference manual to resolve the issue of missing documentation[4].

[3] `https://ap2p.unomaha.edu/`

[4] The new system and user manual can be found here: `https://github.com/potassco/anthem`

5 Ongoing Directions and Expected Achievements

Recursive aggregates Dr. Fandinno and I are currently developing recursive aggregate semantics for DLV analogous to those we created for CLINGO. We have designed a new translation to a many-sorted first-order language for DLV programs, that treats default negation in a different manner. This work suggests that the difference between the treatment of recursive aggregates in CLINGO versus DLV stems from an underlying difference in their treatment of default negation. These new semantics allow us to define strong equivalence not just between a pair of CLINGO programs or a pair of DLV programs, but between a CLINGO program and a DLV program. These are exciting results that provide new insights into the differences between these two major solvers.

Extending ANTHEM with conditional literals More theoretical work needs to be done before I can implement a translation for conditional literals within ANTHEM. First and foremost, the current results need to be extended to io-programs. Additionally, while checking strong equivalence does not require tightness, checking external equivalence does. It is not yet clear to me how the notion of a predicate dependency graph changes in the presence of conditional literals, although first-order dependency graphs are a promising direction to explore [36].

Sets and many-sorted logic in ANTHEM This is the last major task I hope to accomplish as part of my dissertation. The question of integrating our aggregate semantics into MINI-GRINGO and ANTHEM is dependent upon our ability to (partially) axiomatize our notion of standard interpretations. This special class of interpretations makes some assumptions about the behavior of the set sort (for instance, that set membership behaves in a standard way). There is reason to believe this is possible – VAMPIRE natively supports a partial axiomatization of integer arithmetic [35] which we extend to a partial axiomatization of two-sorted standard interpretations by including certain custom axioms in every verification task. The Thousands of Problems for Theorem Provers (TPTP) project has numerous partial axiomatizations of theories compatible with VAMPIRE, including some focusing on set theory [45]. We may be able to use these as a starting point, though the specifics (Figure 1) of our many-sorted domain may be hard to express. In particular, we need to capture the restrictions that

1. the numeral universe $|I|^{s_{int}}$ is a subsort of the program term universe $|I|^{s_{prg}}$,
2. the tuple universe $|I|^{s_{tuple}}$ consists of tuples of program terms, and
3. the set universe $|I|^{s_{set}}$ is the power set of the tuple universe.

There is also the question of how theory extensions (such as set theory) impacts the runtime of VAMPIRE and, consequently, the usability of ANTHEM. Our experiments with ANTHEM and AP2P have already shown that certain programs containing integer arithmetic can be deceptively difficult to verify automatically. For example, ANTHEM struggles to verify the external equivalence of

$$p(X*X) \text{ :- } X = 0..n.$$

and

$$p(X*X) \text{ :- } X = -n..n.$$

under the assumption that n is an integer greater than 0.

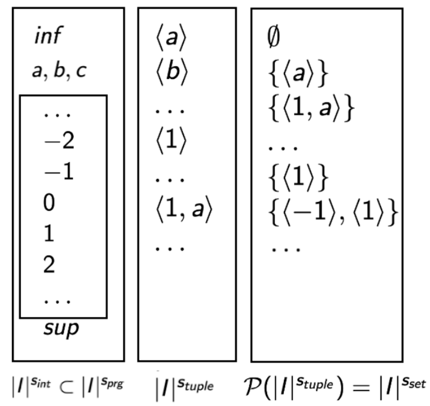


Figure 1: Universes corresponding to the sorts used in standard interpretations [16].

6 Related Work

The Introduction identifies three core challenges to ASP verification: modularity, grounding, and tool support. Within each of these topics, there are a number of studies relevant to the research agenda proposed here.

6.1 Modularity

The notion of external equivalence presented earlier is closely related to modular equivalence for DLP and ELP-functions [33], which are in turn related to LP-functions [26, Section 2]. In these studies, the idea of program modules as composable functions mapping input atoms to output atoms (using hidden auxiliary atoms) has been explored in the context of disjunctive, propositional programs. The *module theorem* [33] provides a compositional semantics for this class of programs.[5] Our work on external behavior of io-programs and modular arguments of correctness uses similar results established for a different class of programs, specifically, programs with variables and arithmetic.

Templates are a construct roughly analogous to program modules, where global (public) predicates are renamed so as to interface with an enclosing program, and local (auxiliary/hidden/private) predicates are renamed with a procedure that (very nearly) guarantees the new identifiers are universally unique [1]. The use of templates promotes code reusability and eases the development of industrial-scale applications [8]. This approach has the additional benefit of being able to test that simple invariants of templates hold in the context of an enclosing program. As an alternative to the re-writing strategy described above, templates can also be defined as higher-order definitions of predicates [12]. This is more in line with the second-order characterization of external behavior given by the *SM* operator.

6.2 Grounding

Within the domain of translational semantics for ASP, a major distinction is between grounding-free approaches, and semantics applied to grounded or propositional programs. A grounding-free translational approach is the basis of this proposal – it benefits from being more general, but suffers some restrictions

[5]Under certain reconfigurations of program modules, the module theorem can apply even when some input, output, or hidden atoms are forgotten [28].

that have been overcome for propositional translations. For example, the limitation of completion semantics to tight programs can be circumvented in the propositional case by adding loop formulas (LF) to the program's (Π) completion ($COMP(\Pi) \cup LF$) [41, Theorem 1].[6] An extension of this theorem to the first-order case exists, but relies on a grounding procedure, namely, an instantiation of $COMP(\Pi) \cup LF$ w.r.t. a finite domain to obtain a propositional theory [10]. Lee and Meng (2011) provide a full generalization of loop formulas to arbitrary first-order formulas – this could be a promising extension to the completion procedure implemented by ANTHEM [36].

6.3 Tool Support

Tools supporting formal verification of ASP programs can be roughly divided into the categories of **testing** based and **proof** based systems. These are complementary approaches, since testing generally cannot provide the same level of assurance but is fast and universally applicable. HARVEY is an ASP-based system for random testing of ASP programs [29]. ASPIDE is an integrated development environment (IDE) supporting unit testing [2, 22]. Similarly to ANTHEM, this supports modular verification by specifying conditions on outputs for program "units" and automatically testing that these conditions are satisfied for certain inputs. However, ANTHEM enables users to specify (possibly infinite) *classes* of inputs rather than a finite set of test cases.

Examples of proof based systems are SELP [9], TABEQL [47], LPEQ [34], and CCT [42]. SELP is closely related to this proposal, since it uses automated reasoning tools to check whether two logic programs are strongly equivalent. One notable consequence of SELP's SAT checking methodology is its ability to find a counterexample, which would be an interesting addition to ANTHEM. SELP differs from ANTHEM in its use of SAT solving instead of theorem proving; a more important difference is that SELP supports disjunctive propositional programs whereas ANTHEM supports normal programs with variables and arithmetic. TABEQL does not translate ASP programs into logical theories, but rather computes equilibrium models of arbitrary propositional theories using tableau calculi for here-and-there. The ASP-to-ASP translation tool LPEQ and its variant DLPEQ produce logic programs whose answer sets (if any) represent counterexamples to the weak or strong equivalence of a pair of programs.[7] Similarly to SELP, these systems accept disjunctive, variable-free logic programs. A more flexible system is CCT, which tests relativised strong equivalence with projection and uniform equivalence.[8] CCT is very similar in spirit to ANTHEM, SELP, and LPEQ given that it relies on a translation to quantified Boolean formulas evaluated by backend solvers. This system is based on a general notion of program correspondence [13], and requires two sets of atoms (a *context* and a *projection set*) which behave similarly to ANTHEM's input and output predicates. Again, ANTHEM is more general: rather than dealing with concrete sets of atoms, ANTHEM operates on classes of inputs defined by first-order assumptions.

7 Conclusions and Future Directions

ASP modules I proposed this idea at the ICLP and LPNMR 2022 doctoral consortiums, and it received encouraging feedback from attendees. Building off of the modular verification methodology discussed earlier, I would like to develop a repository of verified ASP sub-programs ("modules") that provide

[6] The ASSAT tool based on these results uses SAT solvers to compute stable models of propositional logic programs [41].

[7] Two programs are *weakly equivalent* if they have the same answer sets; this is a special case of external equivalence.

[8] Uniform equivalence is a special case of strong equivalence where it is assumed that the context is a set of facts rather than an arbitrary set of rules.

efficient, correct implementations of commonly encountered sub-problems. Each module would be accompanied by a proof of correctness, whose guarantees can be used within an argument of the enclosing program's correctness. This could reduce the effort needed for programming, and some of the labor required to formally prove the correctness of the program. For example, defining the transitive closure of a binary relation is a common task in ASP programming. Integrating a generic module such as

$$
\begin{aligned}
&transitive(X,Y) \text{ :- } edge(X,Y).\\
&transitive(X,Z) \text{ :- } transitive(X,Y), edge(Y,Z).
\end{aligned}
$$

is preferable to re-inventing the wheel with a custom, unverified implementation. In this scenario, the programmer would only have to define the interfaces to and from the module, analogous to the input and output interfaces proposed for disjunctive logic programs [33]. Each program in the repository should have a natural language specification of intended behavior, a proof of correctness, and a description of how to interface with the module. Thus, the idea is similar to that of ASP templates [1], with an emphasis on reusability of the corresponding proofs of correctness. ASP practitioners could submit a (program, specification) pair as a verification challenge, or submit fully verified modules ready for reuse. Besides the transitive closure module, other common, generalizable sub-problems include functional relations (predicates defining a mapping from one set to another) and definitions of grid adjacency (typically used in 2D planning problems like ASPRILO [25]). While I'm still very interested in this task, I've come to realize that such a project is beyond the scope of my dissertation. I now see it as a top priority for post-graduate research.

Discussion The task of verifying ASP programs is complex, but relating them to equivalent theories in the logic of here-and-there and many-sorted first-order logic is a promising approach. Much of my work has focused on investigating this relationship in the presence of advanced language constructs, such as aggregates and conditional literals. These are features on which modern ASP solutions rely heavily - yet they are not even the most sophisticated features offered by modern solvers. Verification techniques for constraint answer set programs, or for programs with optimization statements and theory propagators, are possible future directions of research with great potential.

Another topic of interest is the extension of ANTHEM with alternative backend solvers. Other ASP verification tools support finding counterexamples to program correspondence, which is an interesting and useful ability. An SMT solver like CVC5 [4] might enable ANTHEM to check countersatisfiability and/or generate counterexamples. Furthermore, intuitionistic theorem provers like nanoCOP-i [43] are a natural avenue to explore given ASP's close relationship with the logic of here-and-there. Secondary transformations like completion could be avoided completely if an intuitionistic theorem prover – possibly strengthened with axiom schemata such as (1) – shows itself capable of reasoning effectively within HTA.

Finally, there is the topic of making the verification task accessible enough that ASP practitioners will employ it in the real world. Automation is a great way to promote this – but much more work is required to make complex problems verifiable with reasonable resources. Reusing components of programs and their corresponding proofs of correctness is another way to ease the burden on practitioners. To encourage formal verification in practice, ANTHEM could be integrated as a plugin to an IDE like ASPIDE in addition to behaving as a stand-alone tool. I plan to continue working on such tool-assisted verification strategies in the future.

References

[1] Mario Alviano, Giovambattista Ianni, Francesco Pacenza & Jessica Zangari (2023): *Rethinking Answer Set Programming Templates*. In Martin Gebser & Ilya Sergey, editors: *Practical Aspects of Declarative Languages*, Springer Nature Switzerland, Cham, p. 82–99, doi:10.1007/978-3-031-52038-9_6.

[2] Giovanni Amendola, Tobias Berei & Francesco Ricca (2021): *Testing in ASP: Revisited Language and Programming Environment*. In Wolfgang Faber, Gerhard Friedrich, Martin Gebser & Michael Morak, editors: *Logics in Artificial Intelligence*, Springer International Publishing, Cham, p. 362–376, doi:10.1007/978-3-030-75775-5_24.

[3] Vernon Asuncion, Yin Chen, Yan Zhang & Yi Zhou (2015): *Ordered completion for logic programs with aggregates*. *Artificial Intelligence* 224, p. 72–102, doi:10.1016/j.artint.2015.03.007.

[4] Haniel Barbosa, Clark 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: *Tools and Algorithms for the Construction and Analysis of Systems: 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I*, Springer-Verlag, Berlin, Heidelberg, p. 415–442, doi:10.1007/978-3-030-99524-9_24.

[5] Michael Bartholomew & Joohyung Lee (2019): *First-order stable model semantics with intensional functions*. *Artificial Intelligence* 273, p. 56–93, doi:10.1016/j.artint.2019.01.001.

[6] Pedro Cabalar, Jorge Fandinno & Yuliya Lierler (2020): *Modular Answer Set Programming as a Formal Specification Language*. *Theory and Practice of Logic Programming*, doi:10.1007/978-1-4471-0043-0.

[7] 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 and Practice of Logic Programming* 20(2), p. 294–309, doi:10.1017/S1471068419000450.

[8] Francesco Calimeri & Giovambattista Ianni (2006): *Template programs for Disjunctive Logic Programming: An operational semantics*. *AI Commun.* 19(3), p. 193–206.

[9] Yin Chen, Fangzhen Lin & Lei Li (2005): *SELP – A System for Studying Strong Equivalence Between Logic Programs*. In Chitta Baral, Gianluigi Greco, Nicola Leone & Giorgio Terracina, editors: *Logic Programming and Nonmonotonic Reasoning*, Springer, Berlin, Heidelberg, p. 442–446, doi:10.1007/11546207_43.

[10] Yin Chen, Fangzhen Lin, Yisong Wang & Mingyi Zhang (2006): *First-order loop formulas for normal logic programs*. In: *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'06, AAAI Press, Lake District, UK, p. 298–307.

[11] Keith L. Clark (1978): *Negation as Failure*, pp. 293–322. Springer US, Boston, MA, doi:10.1007/978-1-4684-3384-5_11.

[12] Ingmar Dasseville, Matthias van der Hallen, Gerda Janssens & Marc Denecker (2015): *Semantics of templates in a compositional framework for building logics*. *Theory and Practice of Logic Programming* 15, pp. 681 – 695, doi:10.1017/S1471068415000319.

[13] Thomas Eiter, Hans Tompits & Stefan Woltran (2005): *On solution correspondences in answer-set programming*. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 97–102. Event-place: Edinburgh, Scotland.

[14] François Fages (1994): *Consistency of Clark's completion and the existence of stable models*. *Journal of Methods of Logic in Computer Science* 1, pp. 51–60.

[15] Jorge Fandinno, Zachary Hansen & Yuliya Lierler (2022): *Arguing Correctness of ASP Programs with Aggregates*. In Georg Gottlob, Daniela Inclezan & Marco Maratea, editors: *Logic Programming and Nonmonotonic Reasoning*, Lecture Notes in Computer Science, Springer International Publishing, Cham, p. 190–202, doi:10.1007/978-3-031-15707-3_15.

[16] Jorge Fandinno, Zachary Hansen & Yuliya Lierler (2022): *Axiomatization of Aggregates in Answer Set Programming*. Proceedings of the AAAI Conference on Artificial Intelligence 36(55), p. 5634–5641, doi:10.1609/aaai.v36i5.20504.

[17] Jorge Fandinno, Zachary Hansen & Yuliya Lierler (2024): *Axiomatization of Non-Recursive Aggregates in First-Order Answer Set Programming*. Journal of Artificial Intelligence Research 80, p. 977–1031, doi:10.1613/jair.1.15786.

[18] Jorge Fandinno, Zachary Hansen, Yuliya Lierler, Vladimir Lifschitz & Nathan Temple (2023): *External Behavior of a Logic Program and Verification of Refactoring*. Theory and Practice of Logic Programming 23(4), p. 933–947, doi:10.1017/S1471068423000200.

[19] Jorge Fandinno & Vladimir Lifschitz (2023): *Omega-Completeness of the Logic of Here-and-There and Strong Equivalence of Logic Programs*. Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning 19(1), p. 240–251, doi:10.24963/kr.2023/24.

[20] Jorge Fandinno, Vladimir Lifschitz, Patrick Lühne & Torsten Schaub (2020): *Verifying Tight Logic Programs with anthem and vampire*. Theory and Practice of Logic Programming 5(20), pp. 735–750, doi:10.1017/S1471068403001765.

[21] Jorge Fandinno, Vladimir Lifschitz & Nathan Temple (2024): *Locally Tight Programs*. Theory and Practice of Logic Programming, p. 1–31, doi:10.1017/S147106842300039X.

[22] Onofrio Febbraro, Nicola Leone, Kristian Reale & Francesco Ricca (2013): *Unit Testing in ASPIDE*. In Hans Tompits, Salvador Abreu, Johannes Oetsch, Jörg Pührer, Dietmar Seipel, Masanobu Umeda & Armin Wolf, editors: Applications of Declarative Programming and Knowledge Management, Springer, Berlin, Heidelberg, p. 345–364, doi:10.1007/978-3-642-41524-1_21.

[23] Paolo Ferraris, Joohyung Lee & Vladimir Lifschitz (2011): *Stable models and circumscription*. Artificial Intelligence 175(1), pp. 236–263, doi:10.1016/j.artint.2010.04.011.

[24] Martin Gebser, Amelia Harrison, Roland Kaminski, Vladimir Lifschitz & Torsten Schaub (2015): *Abstract gringo*. Theory and Practice of Logic Programming 15(4–5), p. 449–463, doi:10.1017/S1471068415000150.

[25] Martin Gebser, Philipp Obermeier, Thomas Otto, Torsten Schaub, Orkunt Sabuncu, Van Nguyen & Tran Cao Son (2018): *Experimenting with robotic intra-logistics domains*. Theory and Practice of Logic Programming 18(3–4), p. 502–519, doi:10.1017/S1471068418000200.

[26] Michael Gelfond (2002): *Representing Knowledge in A-Prolog*. Lecture Notes in Computer Science 2408, pp. 413–451, doi:10.1007/3-540-45632-5_16.

[27] 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.

[28] Ricardo Gonçalves, Tomi Janhunen, Matthias Knorr, João Leite & Stefan Woltran (2019): *Forgetting in Modular Answer Set Programming*. Proceedings of the AAAI Conference on Artificial Intelligence 33(01), p. 2843–2850, doi:10.1609/aaai.v33i01.33012843.

[29] Alexander Greßler, Johannes Oetsch & Hans Tompits (2017): *Harvey: A System for Random Testing in ASP*. In Marcello Balduccini & Tomi Janhunen, editors: Logic Programming and Nonmonotonic Reasoning, Springer International Publishing, Cham, p. 229–235, doi:10.1007/978-3-319-61660-5_21.

[30] Zach Hansen & Jorge Fandinno (2023): *Recursive Aggregates as Intensional Functions*. In: Proceedings of Answer Set Programming and Other Computing Paradigms (ASPOCP 2023), pp. 1–22.

[31] Zachary Hansen & Yuliya Lierler (2022): *Semantics for Conditional Literals via the SM Operator*. In Georg Gottlob, Daniela Inclezan & Marco Maratea, editors: Logic Programming and Nonmonotonic Reasoning, Lecture Notes in Computer Science, Springer International Publishing, Cham, p. 259–272, doi:10.1007/978-3-031-15707-3_20.

[32] 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.

[33] T. Janhunen, E. Oikarinen, H. Tompits & S. Woltran (2009): *Modularity Aspects of Disjunctive Stable Models*. Journal of Artificial Intelligence Research 35, p. 813–857, doi:10.1613/jair.2810.

[34] Tomi Janhunen & Emilia Oikarinen (2004): *LPEQ and DLPEQ — Translators for Automated Equivalence Testing of Logic Programs*. In Vladimir Lifschitz & Ilkka Niemelä, editors: Logic Programming and Nonmonotonic Reasoning, Springer, Berlin, Heidelberg, p. 336–340, doi:10.1007/978-3-540-24609-1_30.

[35] Laura Kovács & Andrei Voronkov (2013): *First-Order Theorem Proving and Vampire*. In Natasha Sharygina & Helmut Veith, editors: Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings, Lecture Notes in Computer Science 8044, Springer, pp. 1–35, doi:10.1007/978-3-642-39799-8_1.

[36] J. Lee & Y. Meng (2011): *First-Order Stable Model Semantics and First-Order Loop Formulas*. Journal of Artificial Intelligence Research 42, p. 125–180, doi:10.1613/jair.3337.

[37] Joohyung Lee & Vladimir Lifschitz (2003): *Loop Formulas for Disjunctive Logic Programs*. In Catuscia Palamidessi, editor: Logic Programming, Springer, Berlin, Heidelberg, p. 451–465, doi:10.1007/978-3-540-24599-5_31.

[38] Vladimir Lifschitz (2010): *Thirteen Definitions of a Stable Model*, p. 488–503. Springer, Berlin, Heidelberg, doi:10.1007/978-3-642-15025-8_24.

[39] Vladimir Lifschitz, Patrick Lühne & Torsten Schaub (2020): *Towards Verifying Logic Programs in the Input Language of clingo*. In Andreas Blass, Patrick Cégielski, Nachum Dershowitz, Manfred Droste & Bernd Finkbeiner, editors: Fields of Logic and Computation III, Springer International Publishing, Cham, pp. 190–209, doi:10.1007/978-1-4684-3384-5_11.

[40] Vladimir Lifschitz, David Pearce & Agustín Valverde (2001): *Strongly equivalent logic programs*. ACM Transactions on Computational Logic 2(4), p. 526–541, doi:10.1145/383779.383783.

[41] Fangzhen Lin & Yuting Zhao (2004): *ASSAT: computing answer sets of a logic program by SAT solvers*. Artificial Intelligence 157(1), p. 115–137, doi:10.1016/j.artint.2004.04.004.

[42] Johannes Oetsch, Martina Seidl, Hans Tompits & Stefan Woltran (2009): *Testing Relativised Uniform Equivalence under Answer-Set Projection in the System ccT*. In Dietmar Seipel, Michael Hanus & Armin Wolf, editors: Applications of Declarative Programming and Knowledge Management, Springer, Berlin, Heidelberg, p. 241–246, doi:10.1007/978-3-642-00675-3_16.

[43] Jens Otten (2016): *nanoCoP: A Non-clausal Connection Prover*. In Nicola Olivetti & Ashish Tiwari, editors: Automated Reasoning, Springer International Publishing, Cham, p. 302–312, doi:10.1007/978-3-319-40229-1_21.

[44] David Pearce (2006): *Equilibrium logic*. Annals of Mathematics and Artificial Intelligence 47(1), p. 3–41, doi:10.1007/s10472-006-9028-z.

[45] Geoff Sutcliffe (2017): *The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0*. Journal of Automated Reasoning 59(4), pp. 483–502, doi:10.1007/s10817-017-9407-7.

[46] Miroslaw Truszczynski (2012): *Connecting First-Order ASP and the Logic FO(ID) through Reducts*. Lecture Notes in Computer Science 7265, Springer Berlin Heidelberg, doi:10.1007/978-3-642-30743-0_37. Available at `http://link.springer.com/10.1007/978-3-642-30743-0_37`.

[47] Agustín Valverde (2004): *tabeql: A Tableau Based Suite for Equilibrium Logic*. In Jóse Júlio Alferes & João Leite, editors: Logics in Artificial Intelligence, Springer, Berlin, Heidelberg, p. 734–737, doi:10.1007/978-3-540-30227-8_69.

[48] Mark Wallace (1993): *Tight, consistent, and computable completions for unrestricted logic programs*. The Journal of Logic Programming 15(3), p. 243–273, doi:10.1016/0743-1066(93)90041-E.

Answer Set Counting and its Applications

Mohimenul Kabir
National University of Singapore
School of Computing
Singapore

We have focused on Answer Set Programming (ASP), more specifically, answer set counting, exploring both exact and approximate methodologies. We developed an exact ASP counter, sharpASP, which utilizes a compact encoding for propositional formulas, significantly enhancing efficiency compared to existing methods that often struggle with inefficient encodings. Our evaluations indicate that sharpASP outperforms current ASP counters on several benchmarks. In addition, we proposed an approximate ASP counter, named ApproxASP, a hashing-based counter integrating Gauss-Jordan elimination within the ASP solver, clingo. As a practical application, we employed ApproxASP for network reliability estimation, demonstrating superior performance over both traditional reliability estimators and #SAT-based methods.

1 Introduction

Answer Set Programming (ASP) [22] has emerged as a promising paradigm in knowledge representation and automated reasoning owing to its ability to model hard combinatorial problems from diverse domains in a natural way [3]. Building on advances in propositional SAT solving, the past two decades have witnessed the emergence of well-engineered systems for solving the answer set satisfiability problem, i.e., finding models or answer sets for a given answer set program. In recent years, there has been growing interest in problems beyond satisfiability, such as model counting, in the context of ASP. In this work, we focus on the model counting problem in the context of ASP, known as *answer set counting* problem. There has been growing interest in answer set counting, motivated by applications in probabilistic reasoning and network reliability [19, 1, 12]

2 Background and Problem Statement

An *answer set program P* consists of a set of rules, each rule is structured as follows:

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

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 $\mathsf{Rules}(P)$ and $\mathsf{atoms}(P)$ denote the rules and atoms within the program P. In rule r, the operator "not" denotes *default negation* [5]. 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 $\mathsf{Body}(r)^-$. A rule r is called a *constraint* when $\mathsf{Head}(r)$ contains no atom. A program P is called a *disjunctive logic program* if there is a rule $r \in \mathsf{Rules}(P)$ such that $|\mathsf{Head}(r)| \geq 2$ [2].

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

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 345–350, doi:10.4204/EPTCS.416.34

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 $\forall_{r \in \mathsf{Rules}(P)} M \models r$. The *Gelfond-Lifschitz (GL) reduct* of a program P, with respect to an interpretation M, is defined as $P^M = \{\mathsf{Head}(r) \leftarrow \mathsf{Body}(r)^+ | r \in \mathsf{Rules}(P), \mathsf{Body}(r)^- \cap M = \emptyset\}$ [11]. An 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 $\mathsf{AS}(P)$.

Exact Answer Set Counting [17] Given an ASP program P, the exact answer set counting seeks to count the number of answer sets of P; more formally, the problem seeks to find $|\mathsf{AS}(P)|$.

Approximate Answer Set Counting [18] Given an ASP program P, tolerance parameter ε, and confidence parameter δ, the approximate answer set counting seeks to estimate the number of answer sets of P with a probabilistic guarantee; more formally, the approximate answer set counters returns a count c such that $\mathsf{Pr}[{}^{|\mathsf{AS}(P)|}/_{1+\varepsilon} \leq c \leq (1+\varepsilon) \times |\mathsf{AS}(P)|] \geq 1-\delta$. Our approximate answer set counter invokes a polynomial number of calls to an ASP solver.

Clark's completion [5] or *program completion* is a technique to translate a normal program P into a propositional formula $\mathsf{Comp}(P)$ that is related but not semantically equivalent. Specifically, for each atom a in $\mathsf{atoms}(P)$, we perform the following steps:

1. Let $r_1, \ldots, r_k \in \mathsf{Rules}(P)$ such that $\mathsf{Head}(r_1) = \ldots = \mathsf{Head}(r_k) = a$, then we add the propositional formula $(a \leftrightarrow (\mathsf{Body}(r_1) \vee \ldots \vee \mathsf{Body}(r_k)))$ to $\mathsf{Comp}(P)$.
2. Otherwise, we add the literal $\neg a$ to $\mathsf{Comp}(P)$.

Finally, $\mathsf{Comp}(P)$ is derived by logically conjoining all the previously added constraints. Literature indicates that while every answer set of P satisfies $\mathsf{Comp}(P)$, the converse is not true [21].

3 Related Works

The decision version of normal logic programs is NP-complete; therefore, the ASP counting for normal logic programs is #P-complete [24] via a polynomial reduction [16]. Given the #P-completeness, a prominent line of work focused on ASP counting relies on translations from the ASP program to a CNF formula [21, 14, 15, 16]. Such translations often result in a large number of CNF clauses and thereby limit practical scalability for *non-tight* ASP programs. Eiter et al. [6] introduced $\mathsf{T_P}$-*unfolding* to break cycles and produce a tight program. They proposed an ASP counter called aspmc, that performs a treewidth-aware Clark completion from a cycle-free program to a CNF formula. Jakl, Pichler, and Woltran [13] extended the tree decomposition based approach for #SAT due to Samer and Szeider [23] to ASP and proposed a *fixed-parameter tractable* (FPT) algorithm for ASP counting. Fichte et al. [9, 8] revisited the FPT algorithm due to Jakl et al. [13] and developed an exact model counter, called DynASP, that performs well on instances with *low treewidth*. Aziz et al. [1] extended a propositional model counter to an answer set counter by integrating unfounded set detection. ASP solvers [10] can count answer set via enumeration, which is suitable for a sufficiently small number of answer sets. Kabir et al. [18] focused on lifting hashing-based techniques to ASP counting, resulting in an approximate counter, called ApproxASP, with (ε, δ)-guarantees. Kabir et al. [17] introduced an ASP counter that utilizes a sophisticated Boolean formula, termed the copy formula, which features a compact encoding.

4 Current Progress and Future Goals

We have already engineered two ASP counters: SharpASP [17] and ApproxASP [18]. SharpASP[1] is an exact answer set counter and ApproxASP[2] is an approximate answer set counter.

The principal contribution of SharpASP is to design a scalable answer set counter, without a substantial increase in the size of the transformed propositional formula, particularly when addressing circular dependencies. The key idea behind a substantial reduction in the size of the transformed formula is an alternative yet correlated perspective of defining answer sets. This alternative definition formulates the answer set counting problem on a pair of Boolean formulas (F,G), where the formula F over-approximates the search space of answer sets, while the formula G exploits *justifications* to identify answer sets correctly. We set $F = \mathsf{Comp}(P)$ since every answer set satisfies Clark completion. Note that $\mathsf{Comp}(P)$ overapproximates answers sets of P. We propose another formula, named *copy formula*, denoted as $\mathsf{Copy}(P)$, which comprises a set of (implicitly conjoined) implications defined as follows:

1. (type 1) for every $v \in \mathsf{LA}(P)$, the implication $v' \rightarrow v$ is in $\mathsf{Copy}(P)$.
2. (type 2) for every rule $x \leftarrow a_1, \dots a_k, b_1, \dots b_m, \sim c_1, \dots \sim c_n$ in P, where $x \in \mathsf{LA}(P)$, $\{a_1, \dots a_k\} \subseteq \mathsf{LA}(P)$ and $\{b_1, \dots b_m\} \cap \mathsf{LA}(P) = \emptyset$, the implication ${a_1}' \wedge \dots {a_k}' \wedge b_1 \wedge \dots b_m \wedge \neg c_1 \wedge \dots \neg c_n \rightarrow x'$ is in $\mathsf{Copy}(P)$.
3. No other implication is in $\mathsf{Copy}(P)$.

For each satisfying assignment $M \models \mathsf{Comp}(P)$, we have the following observations:

- if $M \in \mathsf{AS}(P)$, then $\mathsf{Copy}(P)_{|M} = \emptyset$
- if $M \notin \mathsf{AS}(P)$, then $\mathsf{Copy}(P)_{|M} \neq \emptyset$

where $\mathsf{Copy}(P)_{|M}$ denotes the *unit propagation* of M on $\mathsf{Comp}(P)$. We integrate these observations into propositional model counters to engineer an answer set counter.

Within ApproxASP, we present a scalable approach to approximate the number of answer sets. Inspired by approximate model counter ApproxMC [4], our approach is based on systematically adding parity (XOR) constraints to ASP programs to divide the search space uniformly and independently. We prove that adding random XOR constraints partitions the answer sets of an ASP program. When a randomly chosen partition is *quite small*, we can approximate the number of answer sets by simple multiplication. The XOR semantic in answer set programs was initiated by Everardo et al. [7]. In practice, we use a *Gaussian elimination*-based approach by lifting ideas from SAT to ASP and integrating them into a state-of-the-art ASP solver.

Our objective is to develop more efficient answer set counters by integrating specialized ASP counting techniques and advanced preprocessing methods. Furthermore, we are dedicated to enhancing the capabilities of SharpASP, currently limited to handling normal programs, to also support disjunctive answer set programs. In addition, we are eager to explore broader applications of ASP counting to demonstrate its versatility and potential in solving complex problems. We are also eager to extend the counting technique in other theories [20].

5 Some Results

We implemented prototypes of both SharpASP, on top of the existing propositional model counter SharpSAT-TD (denoted as SharpASP (STD) and ApproxASP, on top of ASP solver Clingo. Finally, we

[1] `https://github.com/meelgroup/SharpASP`

[2] `https://github.com/meelgroup/ApproxASP2`

	clingo	ASProb	aspmc+STD	lp2sat+STD	SharpASP (STD)
#Solved	869	188	840	776	**1023**
PAR2	4285	8722	4572	5084	**3373**

Table 1: The performance comparison of SharpASP vis-a-vis other ASP counters in terms of the number of solved instances and PAR2 scores.

		Clingo	DynASP	Ganak	ApproxMC	ApproxASP
Normal	#Instances	1500				
	#Solved	738	47	973	**1325**	1323
	PAR-2	5172	9705	3606	**1200**	1218
Disjunc.	#Instances	200				
	#Solved	177	0	0	0	**185**
	PAR2	1372	10000	10000	10000	**795**

Table 2: The runtime performance comparison of Clingo, DynASP, Ganak, ApproxMC, and ApproxASP on all considered instances.

empirically evaluate their performance against existing counting benchmarks used in answer set counting literature [9, 6, 1].

SharpASP Our extensive empirical analysis of 1470 benchmarks demonstrates significant performance gain over current state-of-the-art exact answer set counters. The result demonstrated is presented in Table 1 and the rightmost column presents the result of SharpASP. Specifically, by using SharpASP, we were able to solve 1023 benchmarks with a PAR2 score of 3373, whereas using prior state-of-the-art, we could only solve 869 benchmarks with a PAR2 score of 4285. A detailed experimental analysis revealed that the strength of SharpASP is that it spends less time in binary constraint propagation while making more decisions compared to off-the-shelf propositional model counters.

ApproxASP Table 2 presents the result of ApproxASP with state-of-the-art answer set counters. ApproxASP performs well in disjunctive logic programs. ApproxASP solved 185 instances among 200 instances, while the best ASP solver clingo solved a total of 177 instances. In addition, on normal logic programs, ApproxASP performs on par with state-of-the-art approximate model counter ApproxMC.

6 Open issues and expected achievements

Model counting, an intractable problem, is classified as #P for normal programs and #co-NP [9] for disjunctive logic programs, presenting significant challenges in developing scalable answer set counters. Our observations indicate that while our engineered counters effectively scale for certain problem types, they underperform for others. The diverse applications of model counting in real-world scenarios further complicate the creation of application-specific ASP counters. Moreover, we have identified instances where existing systems outperform our SharpASP counter. Integrating strengths from these existing counters into SharpASP to enhance its scalability remains a formidable challenge.

References

[1] Rehan Abdul Aziz, Geoffrey Chu, Christian Muise & Peter James Stuckey (2015): *Stable model counting and its application in probabilistic logic programming*. In: *AAAI*, doi:10.5555/2888116.2888198.

[2] 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.

[3] Alex Brik & Jeffrey Remmel (2015): *Diagnosing automatic whitelisting for dynamic remarketing ads using hybrid ASP*. In: *LPNMR*, Springer, pp. 173–185, doi:10.1007/978-3-319-23264-5_16.

[4] 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.

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

[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] Flavio Everardo, Tomi Janhunen, Roland Kaminski & Torsten Schaub (2019): *The return of xorro*. In: *LPNMR*, Springer, pp. 284–297, doi:10.1007/978-3-030-20528-7_21.

[8] Johannes K Fichte & Markus Hecher (2019): *Treewidth and counting projected answer sets*. In: *LPNMR*, Springer, pp. 105–119, doi:10.1007/978-3-030-20528-7_9.

[9] Johannes Klaus Fichte, Markus Hecher, Michael Morak & Stefan Woltran (2017): *Answer Set Solving with Bounded Treewidth Revisited*. In: *LPNMR*, pp. 132–145, doi:10.1007/978-3-319-61660-5_13.

[10] 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.

[11] Michael Gelfond & Vladimir Lifschitz (1988): *The stable model semantics for logic programming*. In: *ICLP/SLP*, 88, pp. 1070–1080.

[12] Susana Hahn, Tomi Janhunen, Roland Kaminski, Javier Romero, Nicolas Rühling & Torsten Schaub (2022): *Plingo: a system for probabilistic reasoning in Clingo based on LP MLN*. In: *RULEML+RR*, Springer, pp. 54–62, doi:10.48550/arXiv.2206.11515.

[13] Michael Jakl, Reinhard Pichler & Stefan Woltran (2009): *Answer-Set Programming with Bounded Treewidth*. In: *IJCAI*, 9, pp. 816–822, doi:10.5555/1661445.1661575.

[14] Tomi Janhunen (2004): *Representing normal programs with clauses*. In: *ECAI*, 16, p. 358, doi:10.5555/3000001.3000077.

[15] Tomi Janhunen (2006): *Some (in) translatability results for normal logic programs and propositional theories*. *Journal of Applied Non-Classical Logics* 16(1-2), pp. 35–86, doi:10.3166/jancl.16.35-86.

[16] Tomi Janhunen & Ilkka Niemelä (2011): *Compact Translations of Non-disjunctive Answer Set Programs to Propositional Clauses*, pp. 111–130. doi:10.1007/978-3-642-20832-4_8.

[17] Mohimenul Kabir, Supratik Chakraborty & Kuldeep S Meel (2024): *Exact ASP Counting with Compact Encodings*. In: *AAAI*, 38, pp. 10571–10580, doi:10.48550/arXiv.2312.11936.

[18] 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.

[19] 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.

[20] Mohimenul Kabir & Kuldeep S Meel (2024): *On Lower Bounding Minimal Model Count*. *ICLP (to appear)*, doi:10.48550/arXiv.2407.09744.

[21] Fangzhen Lin & Yuting Zhao (2004): *ASSAT: computing answer sets of a logic program by SAT solvers*. *Artificial Intelligence* 157(1), pp. 115 – 137, doi:10.5555/777092.777113. Available at `http://www.sciencedirect.com/science/article/pii/S0004370204000578`.

[22] Victor W Marek & Miroslaw Truszczyński (1999): *Stable models and an alternative logic programming paradigm*. *The Logic Programming Paradigm: a 25-Year Perspective*, pp. 375–398, doi:10.1007/978-3-642-60085-2_17.

[23] Marko Samer & Stefan Szeider (2010): *Algorithms for propositional model counting*. *Journal of Discrete Algorithms* 8(1), pp. 50–64, doi:10.1007/978-3-540-75560-9_35.

[24] Leslie G Valiant (1979): *The complexity of enumeration and reliability problems*. *SIAM Journal on Computing* 8(3), pp. 410–421, doi:10.1137/0208032.

Logical Foundations of Smart Contracts

Kalonji Kalala
School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Canada
hkalo081@uottawa.ca

Nowadays, sophisticated domains are emerging which require appropriate formalisms to be specified accurately in order to reason about them. One such domain is constituted of smart contracts that have emerged in cyber physical systems as a way of enforcing formal agreements between components of these systems. Smart contracts self-execute to run and share business processes through Blockchain, in decentralized systems, with many different participants. Legal contracts are in many cases complex documents, with a number of exceptions, and many subcontracts. The implementation of smart contracts based on legal contracts is a long and laborious task, that needs to include all actions, procedures, and the effects of actions related to the execution of the contract. An ongoing open problem in this area is to formally account for smart contracts using a uniform and somewhat universal formalism. This thesis proposes logical foundations to smart contracts using the Situation Calculus, a logic for reasoning about actions. Situation Calculus is one of the prominent logic-based artificial intelligence approaches that provides enough logical mechanism to specify and implement dynamic and complex systems such as contracts. Situation Calculus is suitable to show how worlds dynamically change. Smart contracts are going to be implement with Golog (written en Prolog), a Situation Calculus-based programming language for modeling complex and dynamic behaviors.

1 Introduction

This work is motivated by the increasing amount of investigations around Blockchains [16] and cyber physical systems [18] that are coupled with the necessity of smart contracts. Cyber physical systems are smart devices that are connected together with the purpose of collecting data, processing them, and providing intelligent decisions. A Blockchain is represented as a public ledger that requires a software to be shared by peers and run a business. That shared software representing a legal contract among parties is called smart contract. The implementation of a legal smart contract needs to take in account all the complexity of a contract and its numerous subcontracts. There are many existing approaches used to implement smart contracts. Those approaches present a number of limitations in handling all the complexity of smart contracts and their large number of subcontracts. To solve those limitations, the need of an approach to represent legal smart contracts in a way that allows to verify their correctness becomes evident. Logic-based approaches are emerging to provide mechanisms for reasoning about actions and their effects.

This thesis is using the Situation Calculus, a logic for specifying dynamical systems in artificial intelligence, to specify smart contracts and reason about them. Many logic-based formalisms have been investigated previously to represent smart contracts, such as the Deontic Logic [1], Event Calculus [12], Defeasible Logic [7] and other logic-based approaches in [31]. The idea of specifying smart contracts in the Situation Calculus is mentioned for the first time in [4]. Daskalopoulou's main idea is that monitoring and enforcing smart contracts can be supported in a particular application domain by giving a suitable (and in formal view) representation for agreements between parties that accounts for deviations of the

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 351–357, doi:10.4204/EPTCS.416.35

parties' behavior from their obligations and corrections of such deviations. Hence a form of Deontic Logic combined with a dynamic, temporal logic is advocated for modeling smart contracts. She uses Event Calculus to this end, while mentioning that the Situation Calculus could as well have been used for this purpose. Our thesis embarks on a program of incorporating obligation-producing actions into Situation Calculus to capture and specify smart contracts.

Smart Contracts

A *smart contract (e-contract)* is a piece of software that monitors the correct execution of the legal contracts [2][30][22][3]. A smart contract is a automation of the execution of an agreement. An application's rules and regulations can be digitally facilitated, verified, validated, and enforced using a smart contract, which is an executable code on a blockchain network. Without the involvement of other parties, smart contracts enable legitimate transactions. These transactions can be monitored and maintained irreversible [23]. For blockchain applications[22][6], a smart contract fulfils the need for application-specific verification and validation. A blockchain is a new form of infrastructure that has the potential to profoundly alter the way that individuals transact, communicate, organize, and identify themselves [34][9][32][17]. A blockchain system is composed of a network of computational nodes that share a single data structure (the blockchain) and reach agreement on its current state [5][33].

Smart Contract Formalization

Formal languages are mostly involved in the specification of e-contracts. Both syntax and semantics of formal languages are beneficial in the process of either verificating or validating of e-contrats [11][8]. Furthermore, le formal languages are equipped to handle business vocabularies and rule semantics. To enable users to comprehend high-level logic and accurately express e-contract semantics, effective approaches to solve logic-specification consistency must be developed. Many investigations have been carried out to study the use of different formal languages in the modeling of electronic contracts, formal languages such as Event, Default, Situation and Deontic calculus.

2 Problem Statement

An ongoing open problem in this area of smart contracts is to formally account for them using a uniform and somewhat universal formalism. In other words, the challenge is to give a formal semantics for smart contracts. This thesis proposes logical foundations to smart contracts using the Situation Calculus. Since smart contracts deal with obligations of contracting agents, the problem statement amounts to formalizing the notion of obligation in the Situation Calculus, and subsequently using this formalization to specify smart contracts, execute those specifications and use these specifications to prove properties of the smart contracts.

3 Situation Calculus

The situation calculus [14, 25] is a many-sorted and mostly first order language with equality specifically designed for representing dynamically changing world. We consider a version of the situation calculus with four sorts for actions, situations, time points, and objects other that the first three sorts. **Actions** are first order terms consisting of an action function symbol and its arguments, one of which being the action occurrence time. **Situations** are first order terms denoting finite sequences of actions. They are represented using a binary function symbol do: $do(\alpha, s)$ denotes the sequence resulting from adding the action α to an existing sequence s. The constant S_0 (*initial situation*) denotes the empty sequence $[\,]$. **Time points** are the sequence of real numbers. Finally, **objects** represent domain specific individuals other than actions, situations, and time points. The language has an alphabet with variables and a finite number of constants for each sort, a finite number of function symbols called *action functions*, a finite number of function symbols called *situation independent functions*, a finite number of function symbols called *functional fluents*, a finite number of predicate symbols called *situation independent predicates*, and a finite number of predicate symbols called *predicate fluents*. Predicate fluents represent properties whose truth values vary from situation to situation as a consequence of executions of actions. A predicate fluent is denoted by a predicate symbol whose last argument is a situation term. Functional fluents denote values that vary from situation to situation as a consequence of executions of actions. The language also includes special predicates *Poss*, and $\sqsubset$; $Poss(a, s)$ means that the action a is possible in the situation s, and $s \sqsubset s'$ states that the situation s' is reachable from s by performing some sequence of actions. In contract modelling terms, $s \sqsubset s'$ means that s is a proper subtrace of the contract execution s'. The predicate $\sqsubset$ will be useful in formulating properties of contracts.

The Situation Calculus allows a high level of flexibility in representing dynamic environments, thus making possible flexible and nuanced modeling of real-world scenarios. The formalism of Situation Calculus offers an effective way to describe changes and reason about the consequences of actions inside a system. This makes it possible to express change and action in an efficient manner. In our case, a smart contract represents the complex dynamic domain representing the Situation Calculus.

A dynamic domain (e.g., legal contracts) is axiomatized in the Situation Calculus with axioms which describe how and under what conditions the domain is changing or not changing as a result of performing actions. Such axioms are called *basic action theory* in [24]. They include the following classes of sentences: domain independent foundational axioms for situations; action precondition axioms, one for each action term, stating the conditions of change; successor state axioms, one for each fluent, stating how change occurs; specific axioms for time, stating the occurrence times of actions and start times of situations; unique names axioms for action terms; and axioms describing the initial situation of the domain. In addition to the primitive actions mentioned above, complex actions mimicking Algol-like programming language constructs have been introduced to capture the full expressiveness of application domains. These complex actions are going to be implement with GOLOG [15][10], a Situation Calculus-based programming language for complex and dynamic behaviors. Golog interpreter is written in Prolog.

4 Solution

Since smart contracts deal with obligations of contracting agents, our approach consists in formalizing the notion of obligation in the Situation Calculus. In doing so, we extend a well-known solution by Scherl and Levesque [27] to the classical frame problem for knowledge to obligations. We then use the Situation Calculus with obligations to specify smart contracts as follows. First, we construct logical theories called *basic contractual theories* to formalize legal contracts. Basic contractual theories provide the formal semantics of the corresponding legal contracts. Second, we represent legal contracts as processes in the Situation Calculus; such processes lead to situations where desirable properties hold that logically follow from the basic contractual theory representing those legal contracts. We provide an implementable specification, thus allowing one to automatically check many properties of the specification using an interpreter. We use the interpreter to develop a framework for obligation-based programming which we apply to verify properties of the specified smart contracts.

5 Methodology

In this thesis, we based our research on a methodology proposed in Reiter's version of the Situation Calculus [24, 25] by representing smart contract transaction as Situation Calculus actions whose effects are captured as truth values of Situation Calculus fluents. This approach relies on modelling a smart contract as a mostly first-order theory called *basic action theory* [24], augmented with sentences that account for the embedding of the logic of obligations, the so-called deontic logic, into the Situation Calculus. The resulting theory is the *basic contractual theory* mentioned in Section 4. In addition to four foundational axioms for situations [26, 21] that structure the space of situations, basic contractual theories contain a set of *successor state axioms* which extend Reiter's solution to the frame problem to the embedding of deontic logic into the situation calculus. With the basic contractual theories in hand, we represent smart contracts in Situation Calculus as logic-based programs built using complex actions that macro-expand to a sequence of simple actions. These programs are executed using an interpreter that uses the basic contractual theories as background theories for the purpose of proving properties of the formalized smart contracts.

6 Expected Contributions and Goals

The overall goal of the research that underpins the thesis has been stated in Section 2, namely providing logical foundations that constitute a formal semantics for smart contracts; and, to this end, we choose the Situation Calculus as our logic. The expected specific contributions and goals include the following:

- Exploring existing logic-based approaches for formalizing smart contracts and comparing them.
- Extending the solution by Scherl and Levesque to the classical frame problem for knowledge to obligations in the Situation Calculus.
- Representing complex actions for specifying smart contracts as such actions.
- Specifying and proving properties of smart contracts in the language of the Situation Calculus augmented with obligations.
- Extending the GOLOG interpreter [13], a situation calculus-based programming language for defining complex actions in terms of a set of primitive actions axiomatized in the situation calculus.
- Using GOLOG to develop a framework for obligation-based programming.
- Providing a Prolog implementation of the framework.

7 Current Status of the Research

With reference to sections below steps (2)–(4) and parts of Step (5) have been completed:

1. **Background and Related Work**. Here, we introduce the various definitions of smart contract concepts, the ontology and related formalizations of legal contracts. We introduce the different logical formalisms used in modeling smart contracts, such as Event Calculus, Default Logic, Modal Logic, Deontic Logic and Temporal Logic. Also, we present some important related works on logic-based smart contracts. Finally, we compare the presented logical formalisms based on a number of features.
2. **Formal Preliminaries**. In this chapter, we present the situation calculation as enriched by Reiter in [24]. We present the language of the Situation Calculus, its syntax, and its foundational axioms. We then present the main components of the situation calculus machinery, including the basic actions theories and the regression mechanism used for reasoning about actions. Furthermore, we present Scherl and Levesque's extension of the Situation Calculus to account for knowledge and knowledge-producing actions that explicitly create agent knowledge. Scherl and Levesque's solution to the frame problem for knowledge-producing actions is presented, as this solution forms the departure point of our own solution for accounting for obligations. Finally we summarize the main theoretical and practical results of the Situation Calculus.
3. **Formalization of Obligations**. This chapter extends the solution by Scherl and Levesque to the frame problem for knowledge-producing actions to obligation-producing actions. An obligation-producing action is one which enacts obligations on the part of whoever agent performs it. Both works have their roots in the seminal work of Raymond Reiter who proposed the so-called successor state axioms as a solution to the frame problem for actions that are neither knowledge, nor obligation producing. The specification in this chapter yields intuitive properties that one would expect from obligations. Obligation-producing actions do only affect a newly introduced fluent for capturing the notion of obligation in the Situation Calculus, and no other fluents, except those fluents that are made obligatory by obligation-producing actions. In addition, persistence appears as a consequence of this new setting: if something is obligatory to an agent in a given situation, it remains obligatory everywhere as it should be, unless something contrary to the obligation occurs. We show that Reiter's regression operator for reasoning about actions back to the initial situation is a reasoning mechanism for this setting as well.
4. **Formalization of Smart Contracts**. We use the Symboleo [28] ontology to specify smart contracts. Symboleo is defined as a formal specification language for legal contracts[20]. Based on Event Calculus, it contains axioms to specify its semantics and its syntax is formalized by a grammar [29]. Symboleo generates smart contracts from natural language contract. In order to make an actual legal contract larger, Symboleo includes a contract ontology containing components such as : obligations, powers and state models to apply to the concepts of contracts [19]. We will use the Symboleo ontology and language to formulate smart contract Symboleo programs which will be systematically translated to Situation Calculus specifications in an effort to have implementable specifications.
5. **Obligation-Based Programming Framework**. This could be addressed by extending of GOLOG to an Obligation-Based Programming Framework and writing the Prolog Implementation of the Programming Framework.

References

[1] S. Azzopardi, G.J. Pace & F. Schapachnik (2018): *On observing contracts: Deontic contracts meet smart contracts*. *Frontiers in Artificial Intelligence and Applications* 313(December), pp. 21–30, doi:10.3233/978-

1-61499-935-5-21.

[2] C.D. Clack (2021): *Languages for Smart and Computable Contracts*. *CoRR* abs/2104.03764. arXiv:2104.03764.

[3] A. Daskalopulu (2019): *Logic-Based Tools for the Analysis and Representation of Legal Contracts Logic-Based Tools for the Analysis and Representation of Legal Contracts Aspassia-Kaliopi Daskalopulu Submitted in fulfilment of the requirements for the award of the degree of Docto* (October 2009).

[4] A. Daskalopulu & M.J. Sergot (1997): *The Representation of Legal Contracts*. *AI Soc.* 11(1), pp. 6–17, doi:10.1007/BF02812435.

[5] A. Dorri, S.S. Kanhere, R. Jurdak & P. Gauravaram (2017): *Blockchain for IoT security and privacy: The case study of a smart home*. In: *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)*, IEEE, pp. 618–623, doi:10.1109/PERCOMW.2017.7917634.

[6] C. Elsden, A. Manohar, J. Briggs, M. Harding, C. Speed & J. Vines (2018): *Making sense of blockchain applications: A typology for HCI*. In: *Proceedings of the 2018 chi conference on human factors in computing systems*, pp. 1–14, doi:10.1145/3173574.3174032.

[7] G.K. Giannikis & A. Daskalopulu (2006): *Defeasible Reasoning with e-Contracts*. In: *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Hong Kong, China, 18-22 December 2006*, IEEE Computer Society, pp. 690–694, doi:10.1109/IAT.2006.51.

[8] G. Governatori, F. Idelberger, Z. Milosevic, R. Riveret, G. Sartor & X. Xu (2018): *On legal contracts, imperative and declarative smart contracts, and blockchain systems*. *Artif. Intell. Law* 26(4), pp. 377–409, doi:10.1007/s10506-018-9223-3.

[9] H. Han, R.K. Shiwakoti, R. Jarvis, C. Mordi & D. Botchie (2023): *Accounting and auditing with blockchain technology and artificial Intelligence: A literature review*. *International Journal of Accounting Information Systems* 48, p. 100598, doi:10.1016/j.accinf.2022.100598.

[10] T. Hofmann, T. Niemueller, J. Claßen & G. Lakemeyer (2016): *Continual planning in Golog*. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, 30.

[11] P.R. Krishna & K. Karlapalem (2008): *Electronic Contracts*. *IEEE Internet Comput.* 12(4), pp. 60–68, doi:10.1109/MIC.2008.77.

[12] J.T. de Kruijff & H. Weigand (2019): *Introducing CommitRuleML for Smart Contracts*. In P. Johannesson, B. Andersson & H. Weigand, editors: *Short Paper Proceedings of the 13th International Workshop on Value Modeling and Business Ontologies, VMBO, 2019, Stockholm, Sweden, March 4-5, 2019*, *CEUR Workshop Proceedings* 2383, CEUR-WS.org. Available at `http://ceur-ws.org/Vol-2383/paper14.pdf`.

[13] H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin & R.B. Scherl (1997): *GOLOG: A Logic Programming Language for Dynamic Domains*. *J. Log. Program.* 31(1-3), pp. 59–83, doi:10.1016/S0743-1066(96)00121-5.

[14] J. McCarthy (1963): *Situations, actions, and causal laws*. Technical Report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE.

[15] S. McIlraith & T.C. Son (2001): *Adapting Golog for programming the semantic web*. In: *Fifth International Symposium on Logical Formalizations of Commonsense Reasoning*, pp. 195–202.

[16] A.A. Monrat, O. Schelén & K. Andersson (2019): *A Survey of Blockchain From the Perspectives of Applications, Challenges, and Opportunities*. *IEEE Access* 7, pp. 117134–117151, doi:10.1109/ACCESS.2019.2936094.

[17] D. Mourtzis, J. Angelopoulos & N. Panopoulos (2023): *Blockchain integration in the era of industrial metaverse*. *Applied Sciences* 13(3), p. 1353, doi:10.3390/app13031353.

[18] A. Panarello, N. Tapas, G. Merlino, F. Longo & A. Puliafito (2018): *Blockchain and IoT Integration: A Systematic Survey*. *Sensors* 18(8), p. 2575, doi:10.3390/s18082575.

[19] A. Parvizimosaed (2020): *Towards the Specification and Verification of Legal Contracts*. *28th IEEE International Requirements Engineering Conference (RE'20). IEEE CS.*

[20] A. Parvizimosaed, S. Sharifi, D. Amyot, L. Logrippo & J. Mylopoulos (2020): *Subcontracting, Assignment, and Substitution for Legal Contracts in Symboleo*. In G. Dobbie, U. Frank, G. Kappel, S.W. Liddle & H.C. Mayr, editors: *Conceptual Modeling - 39th International Conference, ER 2020, Vienna, Austria, November 3-6, 2020, Proceedings*, *Lecture Notes in Computer Science* 12400, Springer, pp. 271–285, doi:10.1007/978-3-030-62522-1_20.

[21] F. Pirri & R. Reiter (1999): *Some contributions to the metatheory of the situation calculus*. *Journal of the ACM (JACM)* 46(3), pp. 325–361, doi:10.1145/316542.316545.

[22] M.A. Rahman, M.M. Rashid, M.S. Hossain, E. Hassanain, M.F. Alhamid & M. Guizani (2019): *Blockchain and IoT-Based Cognitive Edge Framework for Sharing Economy Services in a Smart City*. *IEEE Access* 7, pp. 18611–18621, doi:10.1109/ACCESS.2019.2896065.

[23] B. Ramamurthy (2020): *Blockchain in action*. Manning Publications.

[24] R. Reiter (2001): *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, MIT Press, doi:10.7551/mitpress/4074.001.0001.

[25] R. Reiter (1991): *The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression*. In V. Lifschitz, editor: *Artificial and Mathematical Theory of Computation, Papers in Honor of John McCarthy on the occasion of his sixty-fourth birthday*, Academic Press / Elsevier, pp. 359–380, doi:10.1016/b978-0-12-450010-5.50026-8.

[26] R. Reiter (1993): *Proving properties of states in the situation calculus*. *Artificial Intelligence* 64(2), pp. 337–351, doi:10.1016/0004-3702(93)90109-O.

[27] R.B. Scherl & H.J. Levesque (2003): *Knowledge, action, and the frame problem*. *Artificial Intelligence* 144(1-2), pp. 1–39, doi:10.1016/S0004-3702(02)00365-X.

[28] S. Sharifi, A. Parvizimosaed, D. Amyot, L. Logrippo & J. Mylopoulos (2020): *Symboleo: towards a specification language for legal contracts*. In: *2020 IEEE 28th International Requirements Engineering Conference (RE)*, IEEE, pp. 364–369, doi:10.1109/RE48521.2020.00049.

[29] S.S. Sharifi (2020): *Smart Contracts : from Formal Specification to Blockchain Code*.

[30] H. Surden (2012): *Computable contracts*. *UCDL Rev.* 46, p. 629.

[31] P. Tolmach, Y. Li, S. Lin, Y. Liu & Z. Li (2020): *A Survey of Smart Contract Formal Specification and Verification*. *CoRR* abs/2008.02712. arXiv:2008.02712.

[32] J. Xu, C. Wang & X. Jia (2023): *A survey of blockchain consensus protocols*. *ACM Computing Surveys* 55(13s), pp. 1–35, doi:10.1145/3477138.

[33] X. Yue, H. Wang, D. Jin, M. Li & W. Jiang (2016): *Healthcare data gateways: found healthcare intelligence on blockchain with novel privacy risk control*. *Journal of medical systems* 40, pp. 1–8, doi:10.1007/s10916-016-0574-6.

[34] Z. Zheng, S. Xie, H.-N. Dai, X. Chen & H. Wang (2018): *Blockchain challenges and opportunities: A survey*. *International journal of web and grid services* 14(4), pp. 352–375, doi:10.1504/IJWGS.2018.095647.

Commonsense Reasoning-Aided Autonomous Vehicle Systems

Keegan Kimbrell
The University of Texas at Dallas
Richardson, USA
keegan.kimbrell@utdallas.com

Autonomous Vehicle (AV) systems have been developed with a strong reliance on machine learning techniques. While machine learning approaches, such as deep learning, are extremely effective at tasks that involve observation and classification, they struggle when it comes to performing higher level reasoning about situations on the road. This research involves incorporating commonsense reasoning models that use image data to improve AV systems. This will allow AV systems to perform more accurate reasoning while also making them more adjustable, explainable, and ethical. This paper will discuss the findings so far and motivate its direction going forward.

1 Introduction

For both academic and industry research, AV technology has seen incredible advances since the introduction of computer vision-focused systems in the 1980's [3]. Here, this paper will provide some formal definitions for autonomous vehicles that it will use throughout this writing. SAE International defines autonomous vehicles into six different levels based on the level of automation, with level 0 being no automation and level 5 being full driving automation [6]. Despite AV research being a well-explored field, there are still no level 5, or fully autonomous, vehicles. This is largely due to imperfections in computer vision systems and the complexity of more complicated driving tasks that require a human driver to be present. For a safety-critical system, such as AV systems, minor mistakes cannot be afforded. To this end, it is important that the AV system can make safe and rational decisions based on accurate interpretations about its surroundings.

There are several technologies that are used in the perception side of AV systems, such as Light Detection and Ranging (LiDAR) systems and camera-based systems. These systems are coupled with deep learning techniques such as Convolutional Neural Networks (CNNs), which are used to classify sensor data [14]. However, like all machine learning systems, it is always possible for misclassifications to occur due to noise, scenarios outside of the training data, degradation of sensing equipment, and other external factors. Because of this, AV systems should move towards using a hybrid AI system, or AI that combines deep learning with logical reasoning, to help mitigate the failures and shortcomings of solely deep learning-based approaches.

There are two types of systematic thinking proposed by Kahneman in 2011 [11]. The first is "System 1", which is fast, instinctive, and emotional thinking. The second is "System 2", which is slow, deliberative, and logical. For a human driver, we use both systems when we are in a driving scenario. Identifying objects around us and minor driving actions are done quickly using System 1 thinking. However, when we encounter an unfamiliar or dangerous scenario, we use System 2 thinking to determine a safe way to navigate the situation. In an optimal hybrid AV system, fast System 1 tasks such as perception and classification should be handled by deep learning, and slow System 2 tasks should be handled by commonsense reasoning. The reasoning system can also be used to perform a more deliberative analysis of

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 358–364, doi:10.4204/EPTCS.416.36

the sensor data. This is similar to a human driver who realizes that they misinterpreted an object on the road and looks closer to figure out what it is.

Commonsense reasoning is a method for modeling the human way of thinking about the world around us using default rules and exceptions [9]. In the context of driving scenarios, we can understand this as our default understanding of traffic laws and scenarios. For example, if we drive up to a crosswalk, by default, we know that we have to stop when there are pedestrians waiting to cross. However, there may be an exceptional scenario that breaks this rule, such as if we discover that the pedestrians do not actually intend to cross. In this case, a human driver will still stop and slowly think about the situation and confirm that the pedestrians do not want to cross before making the potentially unsafe decision of driving forward. This research is focused on modeling commonsense reasoning and combining it with current AV techniques to create safer and more reasonable autonomous vehicles.

This experiment proposes a framework for improving AV systems by attaching commonsense layers that use image data to provide feedback to the deep learning layers for various tasks. With this approach, we can write commonsense reasoning models that can perform optimizations, safety checks, and explanations for autonomous vehicles. By keeping the commonsense reasoning model in a separate layer, we can even use this approach to improve existing AV systems. Furthermore, this approach is not encumbered by a mandatory and expensive training process. The commonsense reasoning model can be modeled and updated with rules generated from domain experts, allowing us to easily stay up-to-date on new laws, ethical standards, and regulations. Currently, the commonsense model employs collective behaviors, or the actions of nearby vehicles, to determine the state of the road around us.

2 Related Work

There have been many other works that incorporate symbolic reasoning into deep learning, computer vision, and autonomous vehicle models. Suchan et al. explore commonsense reasoning-based approaches such as an integrated neurosymbolic online abduction vision and semantics-based approach for autonomous driving [17, 16]. These techniques are primarily focused on integrating with the perception model using answer set programming (ASP), a nonmonotonic reasoning system using stable models [9, 13]. While their framework is similar, this approach is more decoupled from the vision models, allowing us to show improvements on existing AV models.

Neurosymbolic AI, AIs that integrate symbolic and neural network-based approaches [10]s, have been applied towards autonomous driving as well. For safety-critical systems, such as autonomous driving, neurosymbolic techniques can improve compliance with guidelines and safety constraints [15]. Anderson et al. propose a neurosymbolic framework that incorporates symbolic policies with a deep reinforcement learning model [1]. They assert that this approach can improve the safety of reinforcement learning approaches in safety-critical domains, including autonomous vehicles. These systems are related to this research in the sense that both are using symbolic methods to improve existing deep learning-based systems. However, this research is different in that it is using commonsense reasoning as the proposed symbolic model and that, while it is being used to improve on a deep learning model, it is a different layer that is generated separately. While autonomous vehicles and computer vision technologies are primarily deep learning-based, this approach could be used to improve upon reinforcement learning-based, other non-neural machine learning-based, or even search-based vehicles.

A framework created earlier, AUTO-DISCERN [12], proposes a goal-directed commonsense reasoning ASP system that makes driving decisions based on the observations of the environment. This research is an extension of this approach by creating a commonsense reasoning model that makes safe decisions and reasons over a road scenario. This experiment pushes it farther by incorporating the model with an AV system and using the commonsense model to improve aspects of autonomous driving.

3 Research Goals

The major goal of creating an AV system of higher level autonomy using commonsense reasoning can be broken down into various tasks relating to where in the AV system we inject the reasoning:

- **Perception and Classification**: Use commonsense reasoning and knowledge to model the sensor data and optimize the classifications. We can also inject commonsense reasoning into the training process itself to create a more connected and explainable model.
- **Safe Decision Making**: We can model rules for the AV system so that it will always make intelligent and safe decisions that still move it towards its desired goal. This is important for making an ethical system that complies with traffic laws.
- **Complicated Tasks**: Complicated tasks may be outside of the training data for an AV system, such as navigating a road after the results of a hurricane. We can create reasoning models that can handle multiple unknown scenarios safely. Furthermore, it is easier to model these niche scenarios using reasoning since there is often a strong bias against such scenarios in the existing training data for AV systems, and they are difficult to capture using just deep learning.

Each of these tasks separately will improve the effectiveness of future AV systems, and if success is found in each task, then they can be combined to create an autonomous vehicle with a higher level of autonomy than existing systems.

4 Preliminary Results

The current focus of these recent experiments has been using commonsense reasoning and knowledge to optimize the classifications of the computer vision model through consistency checking (first and second tasks). The system uses a Prolog [5] commonsense reasoning model to check if the classifications being made by the computer vision model are consistent with each other, particularly if the behavior of nearby vehicles is consistent with the current road scenario. For example, if a traffic light at an intersection is red, then vehicles in that lane should be stopped. We define the group actions of nearby vehicles as *collective behaviors*. If the rules about the collective behaviors are not consistent with the observed objects, then the system adjusts the classifications of objects around the AV system to fix the scenario. This system emulates the human process of reasoning about a road situation by observing surrounding vehicles. In this approach, we test over misclassified traffic light colors and unobserved road obstacles.

To accomplish this, the system takes objects from the computer vision model's output and converts them into facts. For example, the following facts represent the information about nearby vehicles and intersections:

```
property(vehicle, Frame, Object_id,
    Action, VelocityX, VelocityY, Rotation,
    Coordinate1X, Coordinate1Y,
    Coordinate2X, Coordinate2Y).
property(intersection, Frame, Object_id,
    Coordinate1X, Coordinate1Y,
```

CARLA Traffic Lights	Metrics			
	Accuracy	Precision	Recall	F-Score
Town 1 100 NPCs Logic	.9632	.9663	.9942	.98
Town 1 100 NPCs Baseline	.479	.6579	.145	.237
Town 1 100 NPCs Combined	.9547	.9297	.9942	.9609
Town 1 200 NPCs Logic	1	1	1	1
Town 1 200 NPCs Baseline	.7634	.5	.4091	.45
Town 1 200 NPCs Combined	.8387	.64	.7272	.6809

Table 1: Results of commonsense reasoning, baseline deep learning, and combined hybrid models for traffic lights. The logic model is only evaluated over frames in which there are collective behaviors, and the baseline and combined models are evaluated over all frames.

```
    Coordinate2X, Coordinate2Y).
    vehicles(Frame, Vehicles).
```

These facts are treated as knowledge about our current scenario. The system also contains another Prolog program that performs commonsense reasoning over road scenarios. These rules define how nearby vehicles, or collective behaviors, should act around traffic lights and obstacles.

```
false_negative_light(Frame):-
    property(intersection, Frame, _, _, _, _, _),
    collective_{up/down/left/right}(Frame).
```

This rule is a basic example of how the system would detect misclassifications about the presence of red traffic lights. The rule will evaluate to true, meaning that the AV system fails to detect a red traffic light (negative) when there actually is one (positive), when the AV system detects a collective behavior moving across an intersection in front of it. This is similar to how a human driver can figure out a traffic light is red even if it appears to be green based on the behavior of nearby vehicles. In addition to rules concerning traffic lights, the commonsense reasoning program also contains rules about how vehicles behave when near obstacles.

The following are some of the results from the experiments performed so far, which show the results of this system when identifying the color of an incoming traffic light and road obstacles using the collective behaviors of nearby vehicles. Both experiments were performed over recorded datasets from the Car Learning to Act simulator (CARLA) [7].

Table 1 shows the accuracy of the logic model and baseline computer vision model when it comes to identifying the color of traffic lights at intersections. The dataset is generated from two different recordings, about a couple minutes long each. Each recording was done using the same CARLA map (Town 1) with two different vehicle population densities (100 and 200). The image data contained inclement weather conditions that were outside of the training data. Due to this, the baseline model struggles to maintain high accuracy when identifying the color of the traffic light.

The accuracy of the commonsense reasoning model is evaluated over eligible frames in the data, meaning images within the data that fulfill the default rules in our model. For these eligible frames,

CARLA Obstacles	Metrics			
	Accuracy	Precision	Recall	F-Score
Town 3.0 Logic	1	1	1	1
Town 3.0 Baseline	.4943	1	.4943	.6615
Town 3.0 Combined	1	1	1	1
Town 3.1 Logic	1	1	1	1
Town 3.1 Baseline	.93	1	.93	.9636
Town 3.1 Combined	1	1	1	1

Table 2: Results of the commonsense reasoning model for obstructions for the logic, baseline, and hybrid combined models.

the commonsense reasoning model maintains high accuracy. When the reasoning model is combined with the baseline model and evaluated over the whole dataset, we can see a significant increase in all metrics over the baseline model. The increase in performance varies, as it depends strongly on how many frames are eligible for the commonsense reasoning to perform a correction. This is why there is a greater increase in accuracy for the first dataset as opposed to the second. Despite this, the experiment so far has demonstrated that this approach is an effective optimizer for this scenario.

The results from Table 2 show a similar result for a different scenario. In this dataset, scenarios are much shorter (around 30 seconds) and are used to evaluate predictions about incoming obstacles that are blocking a lane of traffic. This is a task that deep learning models can struggle with since they rely entirely on image or sensor data. If a large vehicle is completely obstructing the view of the obstacle, the deep learning system will struggle heavily to identify it. This, however, is not an issue for the commonsense reasoning system. The results show that as long as there are vehicles nearby for us to observe, we can always determine an obstacle blocking a lane. The accuracy of the deep learning model depends heavily on how well it can see the obstruction, which is what leads to the results seen in the table.

5 Conclusion and Future Work

While a lot of progress has been made in research for AV technology, we are still far away from achieving a fully autonomous vehicle. This is because of an overreliance on deep learning techniques. Proposed here is a pipeline towards a fully autonomous vehicle by incorporating commonsense reasoning into various aspects of the AV system. The results so far demonstrate the effectiveness of this approach.

This work will be extended by exploring new techniques to improve the applicability and efficiency of this approach. This approach can be improved with evaluations from real-world datasets, such as KITTI or NuScenes [8, 4], the use of more powerful logic technologies, such as answer set programming, and the exploration of efficient ways to construct and invoke commonsense reasoning models. It will also benefit from the consideration of new ways to combine commonsense reasoning into AV systems, such as employing more neurosymbolic-based methods like injecting commonsense into the training of the deep learning model.

Going forward, the techniques shown for autonomous vehicles can be applied to other domains.

These approaches focus on using commonsense reasoning models that use the images from the autonomous vehicle to improve the system. This can be viewed as a form of visual question answering (VQA [2]) and can be applied to other domains. Future work will be about the knowledge extraction and reasoning from images used in this experiment and demonstrate its effectiveness in various applications, including autonomous vehicles.

References

[1] Greg Anderson, Abhinav Verma, Isil Dillig & Swarat Chaudhuri (2020): *Neurosymbolic reinforcement learning with formally verified exploration*. Advances in neural information processing systems 33, pp. 6172–6183, doi:10.48550/arXiv.2009.12612.

[2] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick & Devi Parikh (2015): *Vqa: Visual question answering*. In: Proceedings of the IEEE international conference on computer vision, pp. 2425–2433, doi:10.1109/ICCV.2015.279.

[3] Keshav Bimbraw (2015): *Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology*. In: 2015 12th international conference on informatics in control, automation and robotics (ICINCO), 1, IEEE, pp. 191–198, doi:10.5220/0005540501910198.

[4] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan & Oscar Beijbom (2020): *nuscenes: A multimodal dataset for autonomous driving*. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 11621–11631, doi:10.1109/CVPR42600.2020.01164.

[5] William F Clocksin & Christopher S Mellish (2003): *Programming in PROLOG*. Springer Science & Business Media, doi:10.1007/978-3-642-55481-0.

[6] On-Road Automated Driving (ORAD) Committee (2021): *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles*. SAE international.

[7] 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, doi:10.48550/arXiv.1711.03938.

[8] Andreas Geiger, Philip Lenz, Christoph Stiller & Raquel Urtasun (2013): *Vision meets robotics: The kitti dataset*. The International Journal of Robotics Research 32(11), pp. 1231–1237, doi:10.1177/0278364913491297.

[9] 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.

[10] Pascal Hitzler, Aaron Eberhart, Monireh Ebrahimi, Md Kamruzzaman Sarker & Lu Zhou (2022): *Neurosymbolic approaches in artificial intelligence*. National Science Review 9(6), p. nwac035, doi:10.1093/nsr/nwac035.

[11] Daniel Kahneman (2011): *Thinking, fast and slow*. macmillan.

[12] Suraj Kothawade, Vinaya Khandelwal, Kinjal Basu, Huaduo Wang & Gopal Gupta (2021): *AUTO-DISCERN: autonomous driving using common sense reasoning*. arXiv preprint arXiv:2110.13606, doi:10.48550/arXiv.2110.13606.

[13] Vladimir Lifschitz (2019): *Answer set programming*. 3, Springer Heidelberg, doi:10.1007/978-3-030-24658-7.

[14] Benjamin Quito (2023): *Compare and Contrast LiDAR and Non-LiDAR Technology in an Autonomous Vehicle: Developing a Safety Framework*. Authorea Preprints, doi:10.36227/techrxiv.23528403.v1.

[15] Amit Sheth, Kaushik Roy & Manas Gaur (2023): *Neurosymbolic artificial intelligence (why, what, and how)*. *IEEE Intelligent Systems* 38(3), pp. 56–62, doi:10.1109/MIS.2023.3268724.

[16] Jakob Suchan, Mehul Bhatt & Srikrishna Varadarajan (2020): *Driven by commonsense*. In: *ECAI 2020*, IOS Press, pp. 2939–2940. doi:10.3233/FAIA200463.

[17] Jakob Suchan, Mehul Bhatt & Srikrishna Varadarajan (2021): *Commonsense visual sensemaking for autonomous driving–On generalised neurosymbolic online abduction integrating vision and semantics*. *Artificial Intelligence* 299, p. 103522, doi:10.1016/j.artint.2021.103522.

A Category-Theoretic Perspective on Approximation Fixpoint Theory*

Samuele Pollaci
Vrije Universiteit Brussel
Brussels, Belgium
Katholieke Universiteit Leuven
Leuven, Belgium
Samuele.Pollaci@vub.be

Approximation Fixpoint Theory (AFT) was founded in the early 2000s by Denecker, Marek, and Truszczyński as an abstract algebraic framework to study the semantics of non-monotonic logics. Since its early successes, the potential of AFT as a unifying semantic framework has become widely recognised, and the interest in AFT has gradually increased, with applications now ranging from foundations of database theory to abstract argumentation. The non-monotonic constructive processes that occur in many more areas of computer science, together with their associated semantic structures, can be successfully studied using AFT, which greatly simplifies their characterizations. The goal of my research is to take a step towards the lifting of AFT into a more general framework for constructive knowledge.

1 Introduction and Related Work

Approximation Fixpoint Theory (AFT) was founded in the 2000s by Denecker, Marek, and Truszczyński [14] as an extension of Tarski's fixpoint theory to study the semantics of non-monotonic logics, like default logic (DL), autoepistemic logic (AEL) and logic programming (LP). In recent years, interest in AFT has gradually increased, with applications now ranging from foundations of database theory to abstract argumentation. Motivated by the success of AFT in this wide range of applications, this project aims at laying the foundation for an extention of AFT from a useful tool in the area non-monotonic logics into a general algebraic theory of constructive knowledge.

In the 1980s and 90s, the area of non-monotonic reasoning (NMR) saw fierce debates about formal semantics. In the subareas of DL, AEL and LP, researchers sought to formalize common-sense intuitions about knowledge of introspective agents. The main contribution of AFT was to demonstrate that, by moving to an algebraic setting, the common principles behind the concepts in these languages can be isolated and studied in a general way. This breakthrough allowed results that were achieved in the context of one of these languages to be easily transferred to another [27, 32].

The core ideas of AFT are relatively simple: we are interested in fixpoints of an operator O on a given lattice $\langle L, \leq \rangle$. For monotonic operators, Tarski's theory guarantees the existence of a least fixpoint, which is of interest in many applications. For non-monotonic operators, the existence of fixpoints is not guaranteed; and even if fixpoints exist, it is not clear which would be "good" fixpoints. AFT generalizes Tarki's theory for monotonic operators by making use of a so-called *approximating operator*; this is an operator $A : L^2 \to L^2$, that operates on L^2, and that is monotonic with respect to the precision order $\leq_p$ (defined by $(x,y) \leq_p (u,v)$ if $x \leq u$ and $v \leq y$)). The intuition is that elements of L^2 approximate

*This PhD project is supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G0B2221N), with supervisors Bart Bogaerts and Marc Denecker.

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 365–373, doi:10.4204/EPTCS.416.37

elements of L: $(x,y) \in L^2$ approximates z if $x \leq z \leq y$, i.e. when $x \leq y$, the tuple (x,y) can be thought of as an interval in L. Given such an approximator, AFT defines several types of fixpoints (supported fixpoints, a Kripke-Kleene fixpoint, stable fixpoints, a well-founded fixpoint) of interest. In several fields of non-monotonic reasoning, it is relatively straightforward to define an approximating operator and it turns out that the different types of fixpoints then correspond to existing semantics. In this way, AFT clarifies on the one hand how different semantics in a single domain relate, and on the other hand what the relation is between different (non-monotonic) logics.

Since its early successes, the potential of AFT as a unifying semantic framework has become widely recognised. It has been applied to a multi-agent extension of AEL [29], to Dung's abstract argumentation frameworks (AFs), and abstract dialectical frameworks (ADFs) [25]. It has also been applied to extensions of LP with aggregates [16], with HEX-atoms [1], with higher-order functions [11], and with description logics [22]. It served as the foundation of the causal logic FO(C) [8] and was used to characterize active integrity constraints [4]. In all these domains, the power of AFT allows a variety of complex semantic structures to be characterized by surprisingly simple approximating operators. In comparison to a direct definition of the semantic structures, the AFT approach therefore greatly reduces the risk of error, while significantly simplifying the mathematical study of these structures. Interest in these application areas has also driven the theoretical development of AFT in new directions [22, 11, 5, 7, 17].

The recent broad interest in AFT and the resulting research output provide a unique opportunity to put AFT on the map as a general algebraic theory of constructive knowledge. Throughout computer science, Tarski's fixpoint theory is used because it allows a potentially complex object (the least fixpoint of a monotone operator) to be constructed as the limit of a simple iteration process. In many applications, however, this simple monotone construction process does not suffice. In such cases, it is sometimes possible to devise other, derived operators that do not operate on the basic semantic space but on a more complex space. The core of AFT lies in the study of the general principles that underlie such approximations. Such non-monotone construction processes occur in many more areas of computer science, including formal verification, functional programming, and database theory. The goal of this project is to develop a single unified framework to study them, thereby bringing two important benefits. First, a general theory provides confidence in the correct characterization of the processes and resulting semantics. Non-monotone construction processes can be highly complex and developing a new formalization of such a process from scratch is a difficult, time-consuming and error-prone task. This advantage of AFT has already been convincingly demonstrated for logic programs with aggregates, where direct definitions of the desired semantic structures are highly complex, while the use of AFT requires nothing more than the definition of a (three-valued) truth evaluation function for the aggregates. Second, once the constructive process has been correctly characterized, AFT offers a powerful set of algebraic tools to analyze the process and its limit. For instance, a general toolset for the study of modularity properties was developed, generalizing several known concepts from AEL, DL and LP [28, 30, 31, 32]. As a second example, Truszczyński [28] developed tools for the study of strong equivalence, which immediately transfer to other fields, where such notions are now studied independently [23]. AFT greatly simplifies the characterization and subsequent study of constructive processes and their associated semantic structures. By lifting AFT into a more general framework for constructive knowledge, we bring these benefits to a wide range of application areas in computer science.

2 Scientific Research Goals

My PhD research activity fits into the framework of the AFTACK project (project G0B2221N) supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen. The global goal of this project is to take the next big leap forward for AFT by lifting the results obtained for various specific application domains into a general framework for constructive knowledge. To achieve this, three major types of advancements are needed.

A. General approximation spaces: A construction process consists of a series of approximative objects. In AFT, these are elements (x,y) of the bilattice L^2, which correspond to intervals $[x,y] = \{z \in L \mid x \leq z \leq y\}$ in the original lattice. In various scenarios, intervals are not refined enough and a more general approximation space is needed.

B. General processes: AFT was built for processes typically found in non monotonic reasoning. When moving beyond this scope, the basic concepts behind constructive processes remain the same, but certain key differences nevertheless arise. For instance, while processes in AFT typically construct relations over a given set, domain theory [24] considers processes that simultaneously construct the relations and the set over which they are defined.

C. Analysis of processes: The success of AFT is for a large part due to the rich algebraic toolkit it offers to analyze the defined processes and semantics. In parallel with extending the range of spaces and processes, this toolkit needs to be extended as well.

These three goals are materialized in five concrete research objectives. In my research, I am going to focus primarily on three of them:

1. Approximation Spaces (A). Develop a generalization of AFT where the approximations can be complex mathematical structures, instead of simple intervals of lattice elements. In several applications, the limitation to intervals was recognized as a key limiting factor [11, 22, 6, 3]. The generalisation should be general enough to cover these domains.

2. Recursively defined domains and higher-order functions (A,B). Develop extensions of AFT and domain theory suitable to define recursive higher-order functions and predicates. Preliminary experiments have shown that AFT is unsuitable for recursive definitions of higher-order functions [12]. For monotonic recursively defined functions, a solution is provided in domain theory. The objective is to extend domain theory [24] with the fixpoint notions of AFT to handle non-monotonically defined recursive functions and predicates.

3. Explanations for AFT (C). In many domains, it is not only important to reach the right conclusions, but also to explain why they are correct. For instance, in causality, this is the question of actual causation [20]. In the context of constructive knowledge, this question can be posed as "Why does a property hold in the constructed object?". Achieving explainability is especially important in the light of the EU General Data Protection Regulation, article 22 of which requires that all AI with an impact on human lives needs to be accountable. Therefore, we want to obtain a principled approach towards explanations in AFT, which will immediately be applicable to all logics captured by AFT.

The impact of this project is spread over many different domains. In fact, within each new application domain, valuable lessons can be learnt from AFT, as has been witnessed before, for instance in the context of default logic, weighted argumentation, and active integrity constraints. The largest short-term impact is probably found in Objective 2 (Recursively defined domains and higher-order functions). If successful, this project will on the one hand be a bridge between functional programming and non-monotonic reasoning, and on the other hand, will provide the semantic foundations for a new class of function definitions by lifting the restriction that function definitions need to be monotonic in the definedness order. Finally, explainability is an important topic in many of the application domains of AFT right now. By studying this once, algebraically, we lay the foundations for work on explanations in each of these research areas.

3 Research Methodology

In the following, I break down the three objective presented above into smaller work packages (WP). This subdivision follows the one proposed for the AFTACK project. In particular, WP2 will be the main core of my research activity. At the same time, I will collaborate with other members of the research group on (parts of) the other work packages. More details are provided in Section 4.

3.1 WP1 Approximation Spaces

To develop a notion of an approximation space of a given lattice $\langle L, \leq \rangle$, we need a mathematical structure equipped with a truth order $\leq$ and a precision order $\leq_p$, on which we define a generalization of an approximating operator. It is important that the space is equipped with enough structure to allow the construction of key concepts of AFT, such as the stable operator, whose fixpoints determine the partial stable model semantics and the well-founded semantics [14]. We first tackle this problem for two specific structures. We expect the lessons learned there to prove valuable when finally developing approximation spaces in WP1.3.

WP1.1. We first consider the use of the powerset of L as an approximations space. Based on a preliminary analysis, we conjecture that it is impossible to define a suitable truth order on the set of all subsets S of L. Therefore, we focus on sets “without holes”, formally, sets S such that whenever $x, z \in S$ and $x < y < z$, also $y \in S$. The most important challenge is defining a stable operator in this context, and then studying the resulting semantics.

WP1.2. Weighted Abstract Dialectical Frameworks (wADFs) [10] were originally defined in the context of an arbitrary set with a precision order (a complete partial order) but no truth order. In this work package, we research (i) how to add a truth order to wADFs, and (ii) which restrictions on the respective orders we need in order to generalize stable and well-founded semantics to wADFs.

WP1.3. Now, we are concerned with formally defining the notion of an approximation space. This should be a mathematical structure $\langle \mathcal{L}, \leq, \leq_p \rangle$ of approximations of a lattice $\langle L, \leq$ such that (i) there is a $\leq$-preserving injection from L into $\mathcal{L}$; (ii) we can generalize the definition of an approximating operator $A\colon L^2 \to L^2$ on the bilattice L^2 to approximating operators $A\colon \mathcal{L} \to \mathcal{L}$ on the approximation space $\mathcal{L}$; (iii) we can generalize the key concepts of AFT (such as the stable operator) to $\mathcal{L}$; and (iv) the generalizations cover at least the approximations from WP1.1, and WP1.2, and possibly other extensions of AFT [11, 22].

WP1.4. We investigate what it means for an operator on an approximation space to be stratified or modular. We also investigate which types of fixpoints of the generalized AFT behave well under such stratification [30].

3.2 WP2 Recursively defined domains and higher-order functions

Defining semantics for definitions of higher-order objects is challenging for typed languages and even more so for untyped ones. Semantics for typed and untyped monotone higher-order function definitions are developed in denotational semantics for lambda calculus and functional programming languages [26]; one approach is domain theory [24]. Domain theory was the first approach to provide semantics for the untyped lambda calculus and allows for more complex semantic spaces than AFT, with rich classes of approximate objects, suitable for inductive and coinductive constructions of higher-order objects and infinite data structures. It simultaneously constructs approximations of these complex objects and of the functions operating on them, resulting in a space of self-applicable functions including a fixpoint operator through which recursive definitions are given meaning. In another sense, domain theory is more limited than AFT: it only considers continuous (hence, monotone) operators and definitions. Here, we combine the strengths of both approaches.

WP2.1. We study approximation spaces in a category-theoretic setting. In particular, we investigate which classes of approximation spaces form a Cartesian-closed category. One reason we are interested in this is that a Cartesian-closed category provides a foundation for applying AFT to higher-order definitions, as it provides a systematic construction of approximation domains for higher-order concepts from those of base concepts. We undertake this investigation for different notions of approximations.

WP2.2. Here, we exploit the framework of WP2.1 to develop languages and semantics of non-monotone definitions of higher-order sets and functions. The framework supports this (i) by providing construction of approximation domains for higher-order concepts; (ii) by imposing abstract conditions on language constructs that ensure that they are well-behaved (being morphisms in the category); (iii) by being generic in the underlying order, allowing to combine different orders (e.g., simultaneous definition of sets and functions).The range of notions of approximation space in the framework should offer an enriched spectrum with different trade-offs between precision, mathematical complexity, and computational complexity.

WP2.3. In AFT, the approximator is strictly higher in the set-theoretical hierarchy than the objects on which it operates. A powerful property of domain theory is that this is not the case: the objects it constructs can be applied as functions on all constructed objects, including itself. This property is key to defining semantics of untyped lambda calculus. Dana Scott's construction of such a domain is one of the great achievements in the theory of programming languages, but depends on the $\leq_d$-continuity (and hence, monotonicity) of operators of definitions, a condition we seek to relinquish. In this final work package, we examine whether it is possible to partially lift the limitation of $\leq_d$-monotonicity of domain theory and, at the same time, lift AFT's limitation that prevents its use for self-application.

3.3 WP3 Explanations for AFT

To bring explainability to AFT, we start from the theory of justifications [13]. Like AFT, this is a unifying theory that can characterize semantics of various formalisms. However, it is more specific, in the sense that it does not consider operators on an arbitrary lattice, but is focused on the special case of a powerset lattice. This allows justification theory to build detailed explanations of why each element of a set $X \in L$ belongs to X. Nevertheless, there are strong correspondences between AFT and justification theory, including similar notions of duality. The exact relationship remains largely unexplored.

WP3.1. Instead of starting from an operator, justification theory starts from a "justification frame", from which an operator can be derived. These frames can be nested, resulting in "nested justification systems". In this work package, we research the relationship between the concepts of non-nested justification theory and those of AFT. In particular, we verify the conjecture that for all main "branch evaluations" (a concept from justification theory), the semantics induced by justification theory coincides with the equally named fixpoint(s) in AFT. In other words, this WP provides an embedding of non-nested justification theory into AFT. While this result is valuable in itself, the more interesting question is how to generalize AFT using notions from justification theory, which is what we do next.

WP3.2. In WP3.1, we consider only the operators on powerset lattices induced by justification theory. In this work package, we research (i) how to generalize the notion of justification to work for an approximator over an arbitrary complete lattice, or more general, approximation space, (ii) how to automatically obtain justifications from operators (or rather, from approximators), and (iii) how these obtained justifications translate back to justification theory. While the main focus of this work package is on extending approximation fixpoint theory, and bringing justifications to its various fields of application, the results achieved here also have a significant impact on the theory of justifications itself. Indeed, one consequence of these results would be that we now also obtain justifications for so-called ultimate semantics [15].

WP3.3. As mentioned, justification theory allows justification frames to be nested. This results in an elegant way of capturing the semantics of, for instance, nested least and greatest fixpoint definitions. Now, we research how to achieve the same in AFT. In other words, we develop a suitable notion of nested approximations, where a single "step" of a high-level approximation may include an entire fixpoint construction of a lower-level approximation. In addition, it should be possible to choose, for each of the different levels independently which are the fixpoints of interest.

WP3.4. In the context of justification theory, we recently defined notions of duality, one of which is induced by inverting the truth order. In general, we expect that such notions also show up in AFT and that many properties have interesting dual variants, for instance so-called symmetric approximators would be self-dual. In this work package we perform a complete study of duality in AFT in general: we identify dual properties and study which new results we obtain by exploiting them.

The research goals of the AFTACK project in which my research activity fits are both fundamental and ambitious. It is therefore probable that we will not be able to meet every single one of them. However, this need not be a problem, since each of the individual steps towards the desired goals is in itself innovative and will lead to publications in high-impact conferences and journals. Moreover, the WPs are structured in such a way that there is little interdependence between them. The most central WP is that on approximation spaces (WP1). The research line on explanations (WP3) has a minor dependency on WP1, in the sense that it needs to be studied in the general context of approximation spaces in order to have maximum impact. However, even in the unlikely worst case that the WP on approximation spaces should fail completely, studying the topics in the context of the bilattice of standard AFT still provides novel and interesting results. The research line on recursive functions has the strongest dependency on approximation spaces, in the sense that we expect that the bilattice will not suffice for tackling this topic. Nevertheless, even if we cannot construct approximation spaces that are general enough to meet all of our stated goals for that WP, we can still develop a less ambitious extension of the bilattice that will be enough to allow the work on recursive functions to move forward.

4 Current Status and Future Plan

So far, I have been mainly working on WP2 for AFT for higher-order definitions, specifically WP2.1. and WP2.2.. In particular, a paper on the stable semantics for higher-order logic programming has been accepted for ICLP2024, and another paper on the generalization of approximation spaces for higher-order objects using the tools of Category Theory is under review for LPNMR2024. The latter paper is also contributing to WP1.1 and WP1.3 as it proposes a novel, general notion of possible approximation spaces. Continuing this line of work, I am currently studying the relation between our new research outputs and previous approaches, like [11], both from a theoretic perspective and from a more computational one, regarding the complexity in finding the models of a logic program. As next step, I plan to work towards the completion of both WP1 and WP2, in particular by tackling WP1.2. on wADFs, WP1.4., and WP2.3.. The study of AFT for wADFs may lead the way for the application of AFT to bound-founded ASP [2], assumption-based argumentation [9], weighted argumentation [19], probabilistic argumentation [18], and social argumentation [21].

References

[1] Christian Antic, Thomas Eiter & Michael Fink (2013): *Hex Semantics via Approximation Fixpoint Theory*. In: *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, pp. 102–115, doi:10.1007/978-3-642-40564-8_11.

[2] Rehan Abdul Aziz, Geoffrey Chu & Peter J. Stuckey (2013): *Stable model semantics for founded bounds*. *Theory Pract. Log. Program.* 13(4-5), pp. 517–532, doi:10.1017/S147106841300032X.

[3] Bart 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*, pp. 2686–2693, doi:10.1609/aaai.v33i01.33012686.

[4] Bart Bogaerts & Luís Cruz-Filipe (2018): *Fixpoint semantics for active integrity constraints*. *Artif. Intell.* 255, pp. 43–70, doi:10.1016/j.artint.2017.11.003.

[5] Bart Bogaerts, Joost Vennekens & Marc Denecker (2015): *Grounded fixpoints and their applications in knowledge representation*. *Artif. Intell.* 224, pp. 51–71, doi:10.1016/j.artint.2015.03.006.

[6] Bart Bogaerts, Joost Vennekens & Marc Denecker (2016): *On Well-Founded Set-Inductions and Locally Monotone Operators*. *ACM Trans. Comput. Log.* 17(4), p. 27, doi:10.1145/2963096.

[7] Bart Bogaerts, Joost Vennekens & Marc Denecker (2018): *Safe inductions and their applications in knowledge representation*. *Artif. Intell.* 259, pp. 167–185, doi:10.1016/j.artint.2018.03.008.

[8] Bart Bogaerts, Joost Vennekens, Marc Denecker & Jan Van den Bussche (2014): *FO(C): A Knowledge Representation Language of Causality*. *Theory Pract. Log. Program.* 14(4–5-Online-Supplement), pp. 60–69, doi:10.48550/arXiv.1405.1833.

[9] Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski & Francesca Toni (1997): *An Abstract, Argumentation-Theoretic Approach to Default Reasoning*. *Artif. Intell.* 93, pp. 63–101, doi:10.1016/S0004-3702(97)00015-5.

[10] Gerhard Brewka, Hannes Strass, Johannes Peter Wallner & Stefan Woltran (2018): *Weighted Abstract Dialectical Frameworks*. In: *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*, pp. 1779–1786, doi:10.1609/aaai.v32i1.11545.

[11] Angelos Charalambidis, Panos Rondogiannis & Ioanna Symeonidou (2018): *Approximation Fixpoint Theory and the Well-Founded Semantics of Higher-Order Logic Programs*. Theory Pract. Log. Program. 18(3-4), pp. 421–437, doi:10.1017/S1471068418000108.

[12] Ingmar Dasseville, Matthias van der Hallen, Bart Bogaerts, Gerda Janssens & Marc Denecker (2016): *A Compositional Typed Higher-Order Logic with Definitions*. In: Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA, pp. 14:1–14:13, doi:10.4230/OASIcs.ICLP.2016.14.

[13] Marc Denecker, Gerhard Brewka & Hannes Strass (2015): *A Formal Theory of Justifications*. In: Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings, pp. 250–264, doi:10.1007/978-3-319-23264-5_22.

[14] Marc Denecker, Victor Marek & Mirosław Truszczyński (2000): *Approximations, Stable Operators, Well-Founded Fixpoints and Applications in Nonmonotonic Reasoning*. In Jack Minker, editor: Logic-Based Artificial Intelligence, The Springer International Series in Engineering and Computer Science 597, Springer US, pp. 127–144, doi:10.1007/978-1-4615-1567-8_6.

[15] Marc Denecker, Victor W. Marek & Miroslaw Truszczynski (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.

[16] Marc Denecker, Nikolay Pelov & Maurice Bruynooghe (2001): *Ultimate Well-Founded and Stable Semantics for Logic Programs with Aggregates*. In: Logic Programming, 17th International Conference, ICLP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings, pp. 212–226, doi:10.1007/3-540-45635-X_22.

[17] Marc Denecker & Joost Vennekens (2007): *Well-Founded Semantics and the Algebraic Theory of Non-monotone Inductive Definitions*. In: Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings, pp. 84–96, doi:10.1007/978-3-540-72200-7_9.

[18] Phan Minh Dung & Phan Minh Thang (2010): *Towards (Probabilistic) Argumentation for Jury-based Dispute Resolution*. In: Computational Models of Argument: Proceedings of COMMA 2010, Desenzano del Garda, Italy, September 8-10, 2010, pp. 171–182, doi:10.3233/978-1-60750-619-5-171.

[19] Paul E. Dunne, Anthony Hunter, Peter McBurney, Simon Parsons & Michael J. Wooldridge (2011): *Weighted argument systems: Basic definitions, algorithms, and complexity results*. Artif. Intell. 175(2), pp. 457–486, doi:10.1016/j.artint.2010.09.005.

[20] Joseph Y. Halpern (2016): *Appropriate Causal Models and the stability of Causation*. Rev. Symb. Log. 9(1), pp. 76–102, doi:10.1017/S1755020315000246.

[21] João Leite & João G. Martins (2011): *Social Abstract Argumentation*. In: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011, pp. 2287–2292. Available at http://ijcai.org/Proceedings/11/Papers/381.pdf.

[22] Fangfang Liu, Yi Bi, Md. Solimul Chowdhury, Jia-Huai You & Zhiyong Feng (2016): *Flexible Approximators for Approximating Fixpoint Theory*. In: Advances in Artificial Intelligence - 29th Canadian Conference on Artificial Intelligence, Canadian AI 2016, Victoria, BC, Canada, May 31 - June 3, 2016. Proceedings, pp. 224–236, doi:10.1007/978-3-319-34111-8_28.

[23] Emilia Oikarinen & Stefan Woltran (2011): *Characterizing strong equivalence for argumentation frameworks*. Artif. Intell. 175(14-15), pp. 1985–2009, doi:10.1016/j.artint.2011.06.003.

[24] Dana Scott (1972): *Continuous lattices*. In E. Lawvere, editor: Toposes, Algebraic Geometry and Logic, Lecture Notes in Mathematics 274, Springer Verlag, pp. 97–136, doi:10.1007/BFb0073967.

[25] Hannes Strass (2013): *Approximating operators and semantics for abstract dialectical frameworks*. Artif. Intell. 205, pp. 39–70, doi:10.1016/J.ARTINT.2013.09.004.

[26] Robert D. Tennent (1991): *Semantics of programming languages*. Prentice Hall International Series in Computer Science, Prentice Hall.

[27] Miroslaw Truszczynski (2006): *Strong and uniform equivalence of nonmonotonic theories - an algebraic approach*. Ann. Math. Artif. Intell. 48(3-4), pp. 245–265, doi:10.1007/S10472-007-9049-2.

[28] Miroslaw Truszczynski (2006): *Strong and uniform equivalence of nonmonotonic theories - an algebraic approach*. Ann. Math. Artif. Intell. 48(3-4), pp. 245–265, doi:10.1007/s10472-007-9049-2.

[29] Pieter Van Hertum, Marcos Cramer, Bart Bogaerts & Marc Denecker (2016): *Distributed Autoepistemic Logic and its Application to Access Control*. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, pp. 1286–1292. Available at http://www.ijcai.org/Abstract/16/186.

[30] Joost Vennekens, David Gilis & Marc 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.

[31] Joost Vennekens, Johan Wittocx, Maarten Mariën & Marc Denecker (2007): *Predicate Introduction for Logics with a Fixpoint Semantics. Part I: Logic Programming*. Fundam. Informaticae 79(1-2), pp. 187–208, doi:10.3233/FUN-2007-791-209.

[32] Joost Vennekens, Johan Wittocx, Maarten Mariën & Marc Denecker (2007): *Predicate Introduction for Logics with Fixpoint Semantics. Part II: Autoepistemic Logic*. Fundam. Informaticae 79(1-2), pp. 209–227, doi:10.3233/FUN-2007-791-209.

Hybrid Answer Set Programming: Foundations and Applications

Nicolas Rühling

University of Potsdam, Germany
Institute of Computer Science
An der Bahn 2, 14476 Potsdam
nruehling@uni-potsdam.de

1 Introduction

Answer Set Programming (ASP; [32]) is being increasingly applied to solve problems from the real-world. However, in many cases these problems have a heterogenous nature which requires features beyond the current capacities of solvers like *clingo* [19]. Consider for example the field of configuration [17]; one of the early successful applications of ASP [20, 23]. A configuration problem usually consists of (at least) a partonomy where parts are parameterized by attributes whose values in turn are restricted by constraints. While in simple cases these attributes are discrete, many industrial applications require attributes that range over large numeric domains (eg. precisions in the milimeter range might be needed). Further, calculations over these attributes can be of linear nature (eg. calculating the total weight by summing up the weight of all parts) as well as non-linear (eg. area or volume of an object, inclination of a conveyor belt, etc). Standards ASP solvers like *clingo* quickly reach their limits when dealing with numeric ranges and calculations as they need to explicitly ground all possible values. Apart from this, representing constraints in ASP that go beyond simple arithmetic expressions or aggregations generally requires considerable effort.

Over the last years, hybrid solvers such as *clingcon* [3][1] and *clingo*[DL] [27][2] which make use of dedicated inference methods for certain kinds of constraints over finite integer domains have already been successfully applied to many problems such as train scheduling [2] and warehouse delivery [38]. However, what is still missing, is a solid, semantic underpinning of these systems.

This issue has first been addressed by introducing the Logic of *Here-and-There with constraints* (HT_c; [12]) as an extension of the Logic of *Here-and-There* (HT; [25]) and its non-monotone extension *Equilibrium Logic* [36]. Nowadays, HT serves as a logical foundation for ASP and has facilitated a broader understanding of this paradigm. The idea is that HT_c (and other extensions; see Section 2) play an analogous role for hybrid ASP.

There remain many open questions about these logics regarding their fundamental characteristics as well as their practical use in solvers, ie. how they can guide the implementation. Having a formal understanding of these hybrid logics is also needed to better understand the inherent structure of the (real-world) problems they are applied to, eg. configuration, and to improve their representations in ASP.

[1] https://potassco.org/clingcon
[2] https://github.com/potassco/clingo-dl

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 374–380, doi:10.4204/EPTCS.416.38

2 Background

2.1 Hybrid Solvers

Nowadays, ASP solver *clingo* supports so-called *theory atoms* which allow for foreign inference methods [27, 30] following the approach of SAT *modulo theories* (SMT; [5]). This has greatly facilitated the development of ASP-based special-purpose systems which make use of dedicated inference methods for certain subclasses of constraints such as difference logic and linear programming. The general idea is that some external theory serves as an oracle by certifying some of a program's stable models and has been characterized for *clingo* in [11]. We proceed by giving a quick introduction of some of the hybrid solvers that are part of the POTASSCO suite [3].

The system *clingcon* is a solver for *Constraint Answer Set Programming* (CASP) and extends the input language of *clingo* with linear equations, represented as theory atoms of the form

$$\&\mathtt{sum}\{k_1 * x_1; \dots; k_n * x_n\} \prec k_0 \tag{1}$$

where x_i is an integer variable and $k_i \in \mathbb{Z}$ an integer constant for $0 \leq i \leq n$; and $\prec$ is a comparison symbol such as $<=, =, !=, <, >, >=$. In *clingo*, theory predicates are preceded by '&'.

System *clingo*[DL] has a more restricted syntax which allows for *difference constraints* over integers. This is a subset of the syntax in (1) where theory atoms have the fixed form $\&\mathtt{sum}\{1 * x; (-1) * y\} <= k$ but are rewritten instead as:

$$\&\mathtt{diff}\{x - y\} \mathrel{\mathtt{<=}} k \tag{2}$$

with x and y integer variables and $k \in \mathbb{Z}$.

A third system is *clingo*[LP] [4] which extends *clingo* to solve linear constraints as dealt with in Linear Programming (LP). The syntax is identical to (1) but the domain now ranges over the real numbers. Notably, [11] contains a formal characterization of all three just mentioned systems.

Lastly, a recent addition is system *fclingo* [5] which makes use of *clingcon* to solve ASP modulo conditional linear constraints with founded variables. While in *clingcon* all integer variables need to have a value assigned, *fclingo* adds a notion of undefinedness and foundedness as known from ASP, ie. there needs to be a justification in the logic program if a variable receives a value in an answer set. Further, the conditional aspect of the linear constraints can be seen as a generalization of the concept of aggregates commonly used in ASP. The syntax of *fclingo* accomodates so-called *assignments* which guarantee that a variable only gets assigned a value if all other variables in its definition are itself defined, ie. justified at some other part in the logic program. For instance, the expression

$$\&\mathtt{in}\{y..y\} =: x$$

only assigns the value of y to x if y has been defined by some other rule. Omitting the assignment would permit y and x to take arbitrary values if not defined elsewhere, thereby circumventing the principle of foundedness.

Further systems not developed by POTASSCO include ASP solver *dlvhex* [39] which supports a similar concept of theory atoms as *clingo*. Other CASP systems include *dingo* [28], *mingo* [34] and *ezsmt* [31]. Different from the aforementioned systems, all three rely on translations to non-ASP solvers.

[3] `https://potassco.org/`
[4] `https://github.com/potassco/clingoLP`
[5] `https://github.com/potassco/fclingo`

2.2 The Logic of Here-and-There and Hybrid Extensions

The logics HT and *Equilibrium Logic* nowawadays serve as a logical foundation for (plain) ASP, having brought upon fundamental results such as the notion of *strong equivalence* [33]. The idea of HT is that of two worlds h and t, generally called *here* and *there*. [6] More precisely, an HT-interpretation is a pair $\langle H, T\rangle$ of sets of atoms such that $H \subseteq T$. This gives rise to a three-valued logic where atoms can either be *true*, *false* or *undefined*. A formula φ is *satisfied* or *holds* in a model $\langle H, T\rangle$ in symbols $\langle H, T\rangle \models \varphi$, if it is true in the model, ie. satisfied at the h-world. A model $\langle H, T\rangle$ of a theory Γ is called an *equilibrium model* if (i) it is total, ie. $H = T$, and (ii) for any H' such that $H' \subset T$, $\langle H, T\rangle \not\models \Gamma$. The term equilibrium model was coined in [35] and there is complete agreement between equilibrium models and the stable models of logic programs as defined in [21].

In an attempt to provide a solid, logical foundation for hybrid systems such as the ones introduced in Section 2.1, a number of extensions of HT for incorporating constraints have been introduced.

The Logic of *Here-and-There with constraints* (HT_c; [12]) allows for capturing constraint theories in the non-monotonic setting and has subsequently been extended with aggregate functions over constraint values and variables [9, 10]. In [10] specifications for aggregate functions in terms of HT_c are given based on two different semantic principles. While the semantics given in [18] ensures that aggregate terms are always defined, [22] prohibits so-called *vicious cycles*. We also refer to the former as *Ferraris* and to the latter as *Gelfond-Zhang* (GZ) aggregate semantics.

The Logic of *Here-and-there with lower bound founded variables* ($\mathrm{HT}_{\mathrm{LB}}$; [8]) generalizes the concept of *foundedness* to integer variables. The idea is that variables get assigned the smallest integer value that can be justified. This can be seen as a generalization of plain HT if one regards Boolean truth values as ordered by letting *true* be greater than *false*.

Both of these extensions can be seen as *black-box* approaches in the sense that the constraints are incorporated as special entities whose syntax and satisfaction relations are generally left open. Thus, the intricacies of the hybrid part are mostly unknown from the logic program perspective. Another HT extension with a *white-box* approach of constraints is *ASP*($\mathscr{AC}$) [13] which generalizes logical connectives as a particular case of more general operations on weighted formulas over semirings. In this setting, operators like logical conjunction $\wedge$ become just one more possible operation that can be combined with others, such as addition or multiplication (depending on the underlying semiring). This results in a very expressive and powerful formalism but at the price of a more complex semantics and the requirement of a semiring structure.

Further white-box approaches are based on the incorporation of intensional or non-Herbrand functions in ASP. For instance, [6] added partial intensional functions to a quantified First-Order version of HT [37] and later extended this to sets and aggregates [7].

2.3 Configuration

A wide range of approaches exist for representing and solving configuration problems across various paradigms [29, 26]. In recent years, ASP has emerged as a promising alternative, as evidenced by several applications [20, 16, 23, 24]. Moreover, [15] developed an object-oriented approach to configuration by directly defining concepts in ASP. In the context of interactive configuration, [14] conducted a comparative evaluation of various systems, including the ASP solver *clingo* as well as SAT and CP systems, for their suitability in this context, finding *clingo* to be as capable as any other system.

[6] This is based on Kripke semantics for intuitionistic logic, see [41]

3 Research

My research focuses on the foundations of hybrid ASP with the goal of both understanding better its fundamental properties and exploiting this knowledge to guide and improve solver implementations. Further, as this research is motivated by problems in real-world applications, another goal is to better understand the essence of these problems and how they can be solved using hybrid ASP. More precisely, the objective is to find mathematical or logical formalizations of these problems which subsequently serve as basis for succint but general ASP representations. These two goals are reciprocally beneficial as a deeper insight into real-world problems will make clearer the necessary research directions on the foundational level. In the context of applications, my current focus lies on problems in the realm of (industrial product) configuration.

3.1 Contributions and Future Work

Regarding the theoretical aspects of my research I am currently working on the theoretical foundations of solver *fclingo* with the goal of improving the current implementation. Here, one of the open issues is that current results in HT_c only allow for the use of GZ aggregate semantics (see Sec. 2.2) in *fclingo*. However, we would like to be able to use Ferraris aggregate semantics which guarantee definedness as known from *clingo*. Our current approach here consists of finding a suitable translation between the two semantics.

Another open issue is the formalization of solver *clingo*[DL] by means of logic $\mathrm{HT}_{\mathrm{LB}}$. The concept of assigning a minimal, founded value to integer variables of $\mathrm{HT}_{\mathrm{LB}}$ seems like a natural match with the difference constraints in *clingo*[DL] which are defined as inequalities, thus, generally have multiple valid solutions but only one or a few minimal ones.

On the practical side of my research, preliminary results have been found in application of (plain and hybrid) ASP to configuration problems. In [40] we developed a principled approach to configuration that included a mathematical formalization of configuration problems with an ASP-based solution. We defined a configuration problem in terms of an abstract model and a concrete instantiation. While the model serves as a blueprint for all possible configurations, the instantation represents a solution. This work was accompanied by a corresponding fact format and two ASP encoding, one for *clingo* and one for *fclingo*, which were subsequently made public [7].

A similar but slightly different work has been done in [4] where we developed the COOMSUITE [8], a workbench for experimentation with industrial-scale product configuration problems. The COOMSUITE is built around product configuration language COOM[1] [9] and provides a COOM grammar for parsing, a specialized ASP translator for conversion into facts, two encodings (one for *clingo* and one for *fclingo*) as well as various benchmark sets. The intention is to ease the development of powerful methods able to perform in industrial settings.

Future work here includes the further study of suitable representations for hybrid solver *fclingo*. The current *fclingo* encodings do not necessarily use all features the solver has to offer, eg. undefinedness of numeric variables, but rather leaves this to non-hybrid ASP. The reason for this is that these encodings have been constructed with a plain ASP encoding as base, only modifying the necessary parts. An approach we want to pursue here is to find a logical formalization of configuration problems in terms of HT_c and use this as basic for new encodings which make more natural use of *fclingo*'s features. We

[7] `https://github.com/potassco/configuration-encoding`

[8] `https://github.com/potassco/coom-suite`

[9] COOM is a domain-specific language developed by denkbares GmbH and used in numerous industrial applications

expect that this will not only improve the knowledge representation but also the performance of the solver.

References

[1] *Coom Language*. `https://www.coom-lang.org/`.

[2] D. Abels, J. Jordi, M. Ostrowski, T. Schaub, A. Toletti & P. Wanko (2021): *Train scheduling with hybrid ASP*. *Theory and Practice of Logic Programming* 21(3), pp. 317–347, doi:10.1017/S1471068420000046.

[3] M. Banbara, B. Kaufmann, M. Ostrowski & T. Schaub (2017): *Clingcon: The Next Generation*. *Theory and Practice of Logic Programming* 17(4), pp. 408–461, doi:10.1017/S1471068417000138.

[4] J. Baumeister, K. Herud, M. Ostrowski, J. Reutelshoefer, N. Rühling, T. Schaub & P. Wanko (2024): *Towards Industrial-scale Product Configuration*. In C. Dodaro & V. Martinez, editors: *Proceedings of the Seventeenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'24)*, Lecture Notes in Artificial Intelligence, Springer-Verlag. To appear.

[5] A. Biere, M. Heule, H. van Maaren & T. Walsh, editors (2009): *Handbook of Satisfiability*. *Frontiers in Artificial Intelligence and Applications* 185, IOS Press, doi:10.3233/FAIA336.

[6] P. Cabalar (2011): *Functional answer set programming*. *Theory and Practice of Logic Programming* 11(2-3), pp. 203–233, doi:10.1017/S1471068410000517.

[7] P. Cabalar, J. Fandinno, L. Fariñas del Cerro & D. Pearce (2018): *Functional ASP with Intensional Sets: Application to Gelfond-Zhang Aggregates*. *Theory and Practice of Logic Programming* 18(3-4), pp. 390–405, doi:10.1017/S1471068418000169.

[8] P. Cabalar, J. Fandinno, T. Schaub & S. Schellhorn (2019): *Lower Bound Founded Logic of Here-and-There*. In F. Calimeri, N. Leone & M. Manna, editors: *Proceedings of the Sixteenth European Conference on Logics in Artificial Intelligence (JELIA'19)*, *Lecture Notes in Computer Science* 11468, Springer-Verlag, pp. 509–525, doi:10.1007/978-3-030-19570-0_34.

[9] P. Cabalar, J. Fandinno, T. Schaub & P. Wanko (2020): *An ASP Semantics for Constraints Involving Conditional Aggregates*. In G. De Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín & J. Lang, editors: *Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20)*, IOS Press, pp. 664–671, doi:10.3233/FAIA200152.

[10] P. Cabalar, J. Fandinno, T. Schaub & P. Wanko (2020): *A Uniform Treatment of Aggregates and Constraints in Hybrid ASP*. In D. Calvanese, E. Erdem & M. Thielscher, editors: *Proceedings of the Seventeenth International Conference on Principles of Knowledge Representation and Reasoning (KR'21)*, AAAI Press, pp. 193–202, doi:10.24963/KR.2020/20.

[11] P. Cabalar, J. Fandinno, T. Schaub & P. Wanko (2023): *On the Semantics of Hybrid ASP Systems Based on Clingo*. *Algorithms* 16(4), doi:10.3390/a16040185. Available at `https://www.mdpi.com/1999-4893/16/4/185`.

[12] P. Cabalar, R. Kaminski, M. Ostrowski & T. Schaub (2016): *An ASP Semantics for Default Reasoning with Constraints*. In R. Kambhampati, editor: *Proceedings of the Twenty-fifth International Joint Conference on Artificial Intelligence (IJCAI'16)*, IJCAI/AAAI Press, pp. 1015–1021, doi:10.5555/3060621.3060762.

[13] T. Eiter & R. Kiesel (2020): *ASP($\mathscr{AC}$): Answer Set Programming with Algebraic Constraints*. *Theory and Practice of Logic Programming* 20(6), pp. 895–910, doi:10.1017/S1471068420000393.

[14] A. Falkner, A. Haselböck, G. Krames, G. Schenner, H. Schreiner & R. Taupe (2020): *Solver Requirements for Interactive Configuration.* *Journal of Universal Computer Science* 26(3), pp. 343–373, doi:10.3897/jucs.2020.019.

[15] A. Falkner, A. Ryabokon, G. Schenner & K. Shchekotykhin (2015): *OOASP: Connecting Object-Oriented and Logic Programming*. In F. Calimeri, G. Ianni & M. Truszczyński, editors: *Proceedings of the Thirteenth*

International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'15), *Lecture Notes in Artificial Intelligence* 9345, Springer-Verlag, pp. 332–345, doi:10.1007/978-3-319-23264-5_28.

[16] A. Felfernig, A. Falkner, M. Atas, S. Erdeniz, C. Uran & P. Azzoni (2017): *ASP-based Knowledge Representations for IoT Configuration Scenarios*. In L. Zhang & A. Haag, editors: *Proceedings of the Nineteenth International Configuration Workshop (CONF'17)*, pp. 62–67.

[17] A. Felfernig, L. Hotz, C. Bagley & J. Tiihonen, editors (2014): *Knowledge-Based Configuration: From Research to Business Cases*. Elsevier/Morgan Kaufmann, doi:10.1016/C2011-0-69705-4.

[18] P. Ferraris (2011): *Logic Programs with propositional connectives and aggregates*. *ACM Transactions on Computational Logic* 12(4), pp. 25:1–25:40, doi:10.1145/1970398.1970401.

[19] M. Gebser, R. Kaminski, B. Kaufmann & T. Schaub (2019): *Multi-shot ASP solving with clingo*. *Theory and Practice of Logic Programming* 19(1), pp. 27–82, doi:10.1017/S1471068418000054.

[20] M. Gebser, R. Kaminski & T. Schaub (2011): *aspcud: A Linux Package Configuration Tool Based on Answer Set Programming*. In C. Drescher, I. Lynce & R. Treinen, editors: *Proceedings of the Second International Workshop on Logics for Component Configuration (LoCoCo'11)*, *Electronic Proceedings in Theoretical Computer Science (EPTCS)* 65, pp. 12–25, doi:10.4204/eptcs.65.2.

[21] M. Gelfond & V. Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In R. Kowalski & K. Bowen, editors: *Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP'88)*, MIT Press, pp. 1070–1080, doi:10.1201/b10397-6.

[22] M. Gelfond & Y. Zhang (2014): *Vicious Circle Principle and Logic Programs with Aggregates*. *Theory and Practice of Logic Programming* 14(4-5), pp. 587–601, doi:10.1017/S1471068414000222.

[23] E. Gençay, P. Schüller & E. Erdem (2019): *Applications of non-monotonic reasoning to automotive product configuration using answer set programming*. *Journal of Intelligent Manufacturing* 30, pp. 1407–1422, doi:10.1007/s10845-017-1333-3.

[24] K. Herud, J. Baumeister, O. Sabuncu & T. Schaub (2022): *Conflict Handling in Product Configuration using Answer Set Programming*. In: *Proceedings of the Fifteenth Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'22)*, *CEUR Workshop Proceedings* 3193, CEUR-WS.org. Available at `https://ceur-ws.org/Vol-3193/paper2ASPOCP.pdf`.

[25] 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.

[26] L. Hotz, A. Felfernig, M. Stumptner, A. Ryabokon, C. Bagley & K. Wolter (2014): *Configuration Knowledge Representation and Reasoning*. In Felfernig et al. [17], chapter 6, pp. 41–72, doi:10.1016/b978-0-12-415817-7.00006-2.

[27] T. Janhunen, R. Kaminski, M. Ostrowski, T. Schaub, S. Schellhorn & P. Wanko (2017): *Clingo goes Linear Constraints over Reals and Integers*. *Theory and Practice of Logic Programming* 17(5-6), pp. 872–888, doi:10.1017/S1471068417000242.

[28] T. Janhunen, I. Niemelä & M. Sevalnev (2009): *Computing Stable Models via Reductions to Difference Logic*. In E. Erdem, F. Lin & T. Schaub, editors: *Proceedings of the Tenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'09)*, *Lecture Notes in Artificial Intelligence* 5753, Springer-Verlag, pp. 142–154, doi:10.1007/978-3-642-04238-6_14.

[29] U. Junker (2006): *Configuration*. In F. Rossi, P. van Beek & T. Walsh, editors: *Handbook of Constraint Programming*, chapter 24, Elsevier Science, pp. 837–873, doi:10.1016/s1574-6526(06)80028-3.

[30] 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.

[31] Y. Lierler & B. Susman (2016): *SMT-Based Constraint Answer Set Solver EZSMT (System Description)*. In M. Carro & A. King, editors: *Technical Communications of the Thirty-second International Conference on Logic Programming (ICLP'16)*, *Open Access Series in Informatics (OASIcs)* 52, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, pp. 1:1–1:15, doi:10.4230/OASICS.ICLP.2016.1.

[32] V. Lifschitz (2019): *Answer Set Programming*. Springer-Verlag, doi:10.1007/978-3-030-24658-7.

[33] V. Lifschitz, D. Pearce & A. Valverde (2001): *Strongly equivalent logic programs*. *ACM Transactions on Computational Logic* 2(4), pp. 526–541, doi:10.1145/383779.383783.

[34] G. Liu, T. Janhunen & I. Niemelä (2012): *Answer Set Programming via Mixed Integer Programming*. In G. Brewka, T. Eiter & S. McIlraith, editors: *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning (KR'12)*, AAAI Press, pp. 32–42.

[35] D. Pearce (1997): *A New Logical Characterisation of Stable Models and Answer Sets*. In J. Dix, L. Pereira & T. Przymusinski, editors: *Proceedings of the Sixth International Workshop on Non-Monotonic Extensions of Logic Programming (NMELP'96)*, *Lecture Notes in Computer Science* 1216, Springer-Verlag, pp. 57–70, doi:10.1007/BFb0023801.

[36] D. Pearce (2006): *Equilibrium logic*. *Annals of Mathematics and Artificial Intelligence* 47(1-2), pp. 3–41, doi:10.1007/s10472-006-9028-z.

[37] D. Pearce & A. Valverde (2004): *Towards a First Order Equilibrium Logic for Nonmonotonic Reasoning*. In J. Alferes & J. Leite, editors: *Proceedings of the Ninth European Conference on Logics in Artificial Intelligence (JELIA'04)*, *Lecture Notes in Computer Science* 3229, Springer-Verlag, pp. 147–160, doi:10.1007/978-3-540-30227-8_15.

[38] D. Rajaratnam, T. Schaub, P. Wanko, K. Chen, S. Liu & T. Son (2023): *Solving an Industrial-Scale Warehouse Delivery Problem with Answer Set Programming Modulo Difference Constraints*. *Algorithms* 16(4), doi:10.3390/a16040216. Available at `https://www.mdpi.com/1999-4893/16/4/216`.

[39] C. Redl (2016): *The dlvhex system for knowledge representation: recent advances (system description)*. *Theory and Practice of Logic Programming* 16(5-6), pp. 866–883, doi:10.1017/S1471068416000211.

[40] N. Rühling, T. Schaub & T. Stolzmann (2023): *Towards a formalization of configuration problems for ASP-based reasoning: Preliminary report*. In J. Horcas, J. Galindo, R. Comploi-Taupe & L. Fuentes, editors: *Proceedings of the Twenty-fifth International Configuration Workshop (CONF'23)*, 3509, CEUR Workshop Proceedings, pp. 85–94. Available at `https://ceur-ws.org/Vol-3509/paper12.pdf`.

[41] D. van Dalen (2001): *Intuitionistic Logic*. In D. Gabbay & F. Guenthner, editors: *Handbook of Philosophical Logic*, 3, Springer-Verlag, pp. 225–339, doi:10.1007/978-94-009-5203-4_4.

Autonomous Task Completion Based on Goal-directed Answer Set Programming

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

Task planning for autonomous agents has typically been done using deep learning models and simulation-based reinforcement learning. This research proposes combining inductive learning techniques with goal-directed answer set programming to increase the explainability and reliability of systems for task breakdown and completion. Preliminary research has led to the creation of a Python harness that utilizes s(CASP) to solve task problems in a computationally efficient way. Although this research is in the early stages, we are exploring solutions to complex problems in simulated task completion.

1 Introduction

Task planning for autonomous agents has been an area of interest in recent years as robotics and deep learning have made major advances. Most approaches to task planning involve the use of deep learning models. The most popular approach is deep reinforcement learning, though recent work has used large-language models (LLMs) as well. Deep learning models generally achieve good results, however, they are uninterpretable and often produce flawed answers with no explanation. Much work has been done to improve the explainability of deep learning models, however, they remain untrustworthy.

A better solution is to use logic programming. Logic programming is a programming paradigm based primarily on the calculation of Horn clauses through the process of entailment. The most common logic programming language is Prolog, though most logic programming languages consist of a Prolog-like collection of facts and rules. One advantage of logic programs is that they are inherently interpretable and their errors can be logically understood and solved. The research proposed in this paper involves using answer set programming to complete tasks in a simulated environment. This will hopefully result in autonomous task planning that is both robust and trustworthy.

2 Background and Relevant Literature

As autonomous agents become more ubiquitous, the focus has turned to their ability to complete complex tasks in the real world, converting high-level instructions (like "fold laundry") to executable plans ("walk to clothes", "grab clothes", etc.). Autonomous task completion can mean anything from unmanned vehicles navigating from one point to another to robotic kitchen assistants designed to make certain foods. For the most part, modern autonomous systems use deep learning models to accomplish this [11]. This commonly takes the form of deep reinforcement learning and more recently LLMs. Deep learning has achieved excellent results on complex problems. However, most deep learning systems are black boxes that lack explainability and interpretability. This is especially dangerous given how dependent deep learning algorithms are on the (often flawed) data they are trained on. This makes it difficult to trust that their answers are correct and unbiased, as explored in DARPA's explainable AI retrospective [6].

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 381–389, doi:10.4204/EPTCS.416.39

This is important in critical systems, such as hospital diagnoses or military applications, where it has to be quickly apparent whether a model is correct or not. LLMs are vulnerable to "hallucination", where they provide incorrect responses that are statistically likely [9]. Additionally, they can be "jailbroken" to respond outside of the bounds they were designed for [2]. In the specific arena of autonomous task completion, LLMs struggle with making task breakdowns that are both correct and executable [10]. Deep learning as a whole is well explained in other high-quality survey papers [3]. This paper will focus more on the importance of explainability, which is often neglected in deep learning models.

Inductive Logic Programming (ILP) is a form of machine learning that codifies its learning in the form of first-order logic. Ever since the term was defined in 1991 [12] as the "intersection of Logic Programming and Machine Learning", ILP has served to solve machine learning problems. ILP can get results rivaling deep learning models while being inherently interpretable and explainable [19]. Recent advances in ILP, such as the FOLD family of algorithms, demonstrate that complex data can be represented in small logic programs using default rules. A more detailed description of default rules and the FOLD family of incremental learning algorithms can be found in the paper by Gupta et al. [7]. The research mentioned above uses a type of logic programming called Answer Set Programming (ASP). Unlike Prolog-based logic programming, which generates a true or false answer for a queried predicate, ASP is used to generate all entailable rules from a knowledge base. This collection is called an answer set. This can be used to generate "multiple worlds" where different answer sets are true.

Traditional ASP, like in Clingo [4], executes an answer set program through the use of a SAT solver and grounding. Grounding involves the generation of the program with all variables substituted with constants in the program. A disadvantage of this approach is that grounding is not always guaranteed to be feasible, which can leave some programs with no ASP solution. The s(CASP) system [1] solves this problem by performing a top-down goal-oriented search which eliminates the need for grounding. This advantage makes s(CASP) well-suited to the representation of complex world states and provides an advantage over other ASP systems [5].

One of the biggest weaknesses of the ILP approach to solving problems is the need for background information and 'program templates'. Program templates are a layout of how the generated information should look in the context of the logic program. A domain expert must provide this program template and explicitly logic program-based background knowledge for most ILP. Thus, for trivial examples, it would be just as easy to include the final found rules in the knowledge base at the start. Additionally, while ILP programs perform very well on data that can be represented in a logic program, logic programs have a difficult time representing complex data. These weaknesses can be overcome through the use of traditional machine learning algorithms to supplement a logic program. This approach increases explainability while utilizing the benefits of deep learning and other machine learning models, such as in the paper by Rajasekharan et al. that uses an s(CASP) knowledge base to constrain an LLM into providing more reliable results [14]. Other examples exist of using some form of knowledge base to improve deep learning algorithms [17] [8], but the use of logic programming to augment other algorithms merits further exploration.

3 Methodology

The research outlined in this paper seeks to explore the use of an s(CASP) knowledge base for autonomous task completion in a simulated virtual environment. To test our system, we use the VirtualHome simulator (shown in Figure 1) as a playground for our s(CASP) agent to perform tasks in. VirtualHome allows for multiple agents to operate in a variety of simulated apartments, and provides a

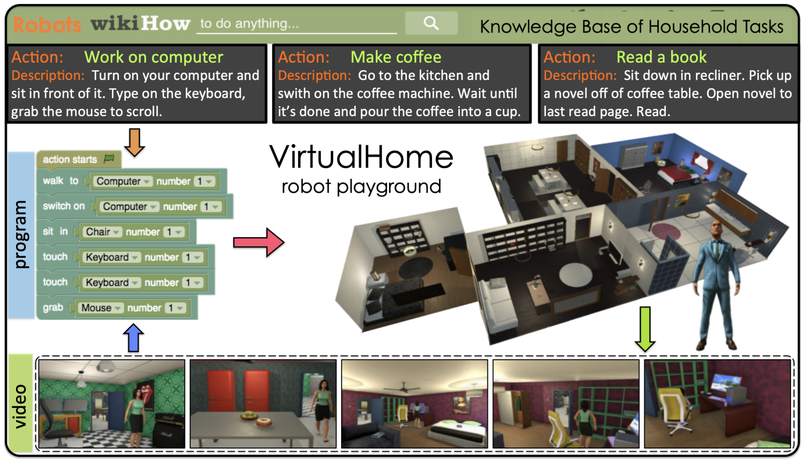


Figure 1: An image showcasing the VirtualHome simulator and an example of its associated task instructions [13].

large database of high-level task breakdowns into step-by-step instructions. This simulation proved to be especially useful for our research because it has a "mid-level" control scheme. This means that we can give the agent commands like "grab remote" rather than dealing with the details of actual movement ("move left foot 3 inches forward", "rotate right arm 45 degrees at the elbow joint", etc) that would be more appropriate for a detailed robotic controller.

The primary goal of this research is to achieve reasonably accurate task completion using goal-directed answer set programming. The end system would have a high level of explainability for decision-making, where the results are trustworthy and could be diagnosed if in error. We wish to further prove that even the very high-quality deep learning systems in use today could be augmented through the use of logic programming. Using logic in this way moves toward general artificial intelligence. Using s(CASP) to simulate how humans can perform common-sense logical interactions with the world brings us closer to reasoning AI.

An additional goal of this research is to make s(CASP) easier to use with simulators. A notable weakness of s(CASP) is that it does not have a Python API, which makes it difficult to run in line with other forms of machine learning. The software engineering goal of this research is to create a "harness" for using s(CASP) in Python for interactions with simulators, as shown in Figure 2.

3.1 Status

Although this research is still in an early stage, there have already been promising results in producing executable actions for small-scale real-world tasks. Using the Python harness mentioned above, the simulated VirtualHome environment can be instantiated and transformed into an s(CASP) representation of the world state:

```
% With Time
current_time(1).
type(livingroom100, livingroom).
type(remotecontrol1, remotecontrol).
off(remotecontrol1, 1).
inside([inside(remotecontrol1, livingroom100),
```

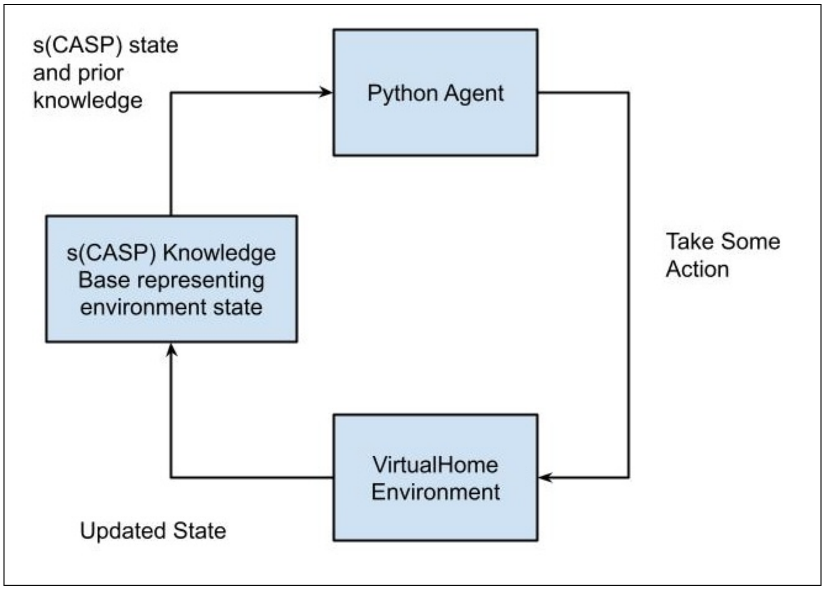


Figure 2: A diagram showing the high-level functionality of the Python harness for s(CASP). The Python harness can perform actions in the VirtualHome environment, and then convert the state of the environment to s(CASP) facts. These facts can then be used to inform the next action of the agent.

```
        inside(character0, livingroom100)], 1).
% Without time
type(livingroom100, livingroom).
type(remotecontrol1, remotecontrol).
off(remotecontrol1).
inside([inside(remotecontrol1, livingroom100),
        inside(character0, livingroom100)]).
```

The above example represents a world state containing a single turned-off remote control sitting in a living room at time 1. The Python harness keeps track of a discretized world time where each action taken by the agent represents a step forward in time, however the addition of time greatly increases the complexity of the world state s(CASP) program. Using time naively in this manner results in intractable programs which loop over infinite time, and so when representing the world state we use the latter example where timestamps are not provided in the state facts. Even without the use of time, this representation of the world state easily grows to encompass a large amount of facts. The complexity of generating an answer set that accounts for all of these facts and possible worlds quickly becomes a computational obstacle. For testing purposes, the Python harness has a small-scale simulation environment built in. Still, the goal remains to execute plans in realistic environments.

To represent and complete tasks we treat task completion as a planning problem. We represent each task as a final state (i.e. if the task was to grab a remote control, the final state would include holds(remotecontrol)) and then formulate actions to reach that final state. The added complexity to this comes from the incorporation of the simulated world state when starting from an initial state. We use the following s(CASP) rules for the task planning problem:

```
% Planning
```

```
% Get the initial state of items close to the character
initial_state(List) :- close_to_character(List).

% Find a set of actions to reach the final state
transform(FinalState, Plan) :-
    initial_state(State1),
    transform(State1, FinalState, [State1], Plan).
transform(State1, FinalState,_,[]) :- subset(FinalState, State1).
transform(State1, State2, Visited, [Action|Actions]) :-
    choose_action(Action, State1, State2),
    update(Action, State1, State),
    not member(State, Visited),
    transform(State, State2, [State|Visited], Actions).

% We choose an action to take
choose_action(Action, State1, State2) :-
    suggest(Action, State2), legal_action(Action, State1).
choose_action(Action, State1, _) :-
    legal_action(Action, State1).
suggest(walk(X), State) :- member(close(X), State).

% Check if an action is legal given the state
legal_action(walk(X), State) :-
    type(X, Y), Y \= character, not member(close(X), State).

% Update state
update(walk(X), State, [close(X) | State1]) :-
    update_walking(X, State, State, [], State1).

% Tasks
complete_task(walk_to_remote, P) :-
    type(Remote, remotecontrol), transform([close(Remote)], P).
```

These rules are a small representative subset of the rules used to generate actions to complete a task. In this very simple example, the task is to walk towards a remote control, which can be easily accomplished by the program. Using this knowledge base we can also achieve some inference. Given a final state where the agent is holding something, using the s(CASP) knowledge base constraints the agent can intuit that it first needs to walk to the item before attempting to pick it up.

A serious problem with this inference, however, is that in sufficiently large environments it becomes too long to calculate (at least over twelve hours without concluding). For example, in the small-scale testing simulation that contains only six items, a plan for "grab the remote" can result in the agent walking to every other object in the room before walking to the remote to grab it. In addition to that solution being inefficient, it takes impossibly long in a real environment with nearly 500 objects. This problem can be solved by adding the rule `suggest(walk(X),State) :-member(holds(X),State), not member(close(X),State).`, however the same then must be done for any other state requiring closeness as a prerequisite. This decreases the value of logical inference and increases the rules required

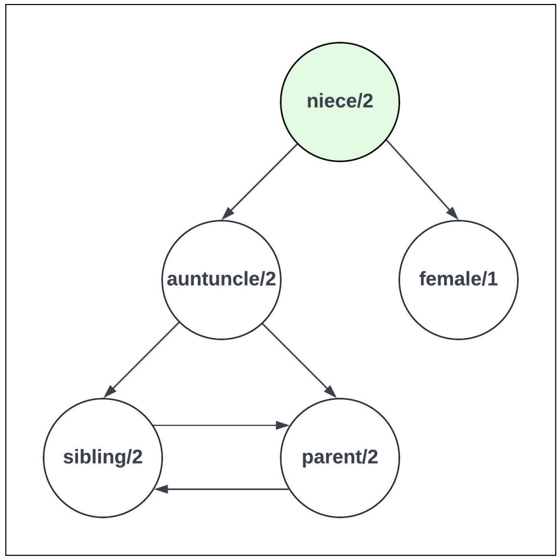


Figure 3: An example dependency graph for a family tree program where niece is the queried rule.

for simple task planning. The limitation remains computation time.

3.2 Preliminary Results

We have made significant strides in reducing the impact of computation time on the program. To reduce computation time, we implemented a dynamic dependency graph that is used to remove facts and rules that are not relevant to the query. For example, given the following knowledge base:

```
parent(tony, abe).
parent(tony, jill).
parent(abe, sarah).

male(tony).
male(abe).
female(jill).
female(sarah).

parent(Parent, Child) :- sibling(X, Child), parent(Parent, X).
grandparent(Grandparent, Child) :-
    parent(Grandparent, Parent), parent(Parent, Child).
sibling(X,Y) :- parent(Parent, X), parent(Parent, Y), X\=Y.
auntuncle(AU,N) :- sibling(AU, Parent), parent(Parent, N).
niece(Niece, AU) :- auntuncle(AU, Niece), female(Niece).
```

Table 1: Table of computational time in seconds for three tasks completed by the s(CASP) agent both optimized and unoptimized.

Task	Unoptimized Time to Complete (s)	Dependency Graph Optimized Time to Complete (s)
Grab Remote Control	13925.14	0.55
Grab Remote Control and Shirt	608.28	0.71
Grab Cell Phone and Sit on Couch	1771.21	0.64

Generating a dependency graph for `?- niece(X,Y).` produces the graph in Figure 3. Using the dependency graph the Python harness can simplify the above knowledge base, removing the male/1 and grandparent/2 predicates entirely. In a program of this size, the computational savings of such optimization is negligible. However, preliminary research has shown a significant time saving in the real-world environment. Table 1 demonstrates the time savings of using the dependency graph to prune the knowledge base for the specific task being accomplished on three semi-simple tasks that take one to four actions to fulfill. The computational time can be reduced from nearly thirty minutes to a fraction of a second using this approach, allowing for continued research into more complex tasks.

3.3 Open Issues and Expected Achievements

Right now, the biggest issues facing this research concern the representation of the s(CASP) knowledge base. There are several outstanding questions.

Representing a Complex Real-World State Representing a simulation of any reasonable size leads to an exponential increase in the number of facts available in the world state. In addition to these facts, there also needs to be a set of rules adequate to perform tasks in the environment. This produces answer sets that are intractable to generate. The use of a dependency graph to pare down the knowledge base allows us to perform more complicated tasks, however there can be more optimization.

Another solution that will be explored is to keep groups of state facts and rules in different programs. The creation of modules that correspond to various tasks or locations would allow for faster calculation of relevant queries. This follows the human logic that one likely does not need their cooking knowledge if, for example, they need to walk their dog.

The Passage of Time As mentioned above, the use of time in the knowledge base provides complications related to the ostensibly infinitely divisible nature of time (as posited by the famous Greek philosopher Zeno). This is a known problem with representing continuous time in logic programming and would require the inclusion of event calculus [18].

Large-scale Learning As deep learning and its applications for real-world task completion are already well explored, the value of this research lies in seeing how complex problems that the s(CASP) task planner can solve can get. To that end, explanation-based learning is a promising paradigm that would allow for generalized knowledge from a small number of examples [16] and works well with answer set programming.

Likely, s(CASP) by itself cannot encode all of the complexities of a real environment and remain tractable. Once that point is reached, there would still be benefits in combining s(CASP) with more traditional machine learning (and newer deep learning, such as LLMs) to improve performance in the former and explainability of the latter. We hope to leverage databases of task instructions and breakdowns, such as those provided by VirtualHome or ALFRED [15], to improve the performance of the s(CASP) agent at scale.

We expect to be able to answer these questions in a unified way to facilitate task completion in complex environments using s(CASP). Although solutions to these problems may always become intractable at certain levels of fidelity, there is valuable knowledge to be gained along the way.

4 Conclusion

In conclusion, this line of research could open up a broad number of solutions for challenging ILP problems. Simply creating a Python framework for the use of s(CASP) with simulated environments is an advancement for s(CASP), as it is currently lacking a Python API. Using the intersection of ILP and traditional machine learning is promising for improving the explainability and reliability of task-completing autonomous agents.

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] Patrick Chao, Alexander Robey, Edgar Dobriban, Hamed Hassani, George J. Pappas & Eric Wong (2023): *Jailbreaking Black Box Large Language Models in Twenty Queries*. arXiv:2310.08419.

[3] Shi Dong, Ping Wang & Khushnood Abbas (2021): *A survey on deep learning and its applications*. *Computer Science Review* 40, p. 100379, doi:10.1016/j.cosrev.2021.100379. Available at `https://www.sciencedirect.com/science/article/pii/S1574013721000198`.

[4] M. Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2014): *Clingo = ASP + Control: Preliminary Report*. *ArXiv* abs/1405.3694, doi:10.48550/arXiv.1405.3694. Available at `http://arxiv.org/abs/1405.3694`.

[5] 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.

[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] Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su & Jun Zhu (2023): *Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications*. arXiv:2211.08064.

[9] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin & Ting Liu (2023): *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*. arXiv:2311.05232.

[10] 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`.

[11] 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.

[12] Stephen Muggleton (1991): *Inductive logic programming*. *New Generation Computing* 8(4), p. 295–318, doi:10.1007/bf03037089.

[13] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler & Antonio Torralba (2018): *VirtualHome: Simulating Household Activities via Programs*. arXiv:1806.07011.

[14] Abhiramon Rajasekharan, Yankai Zeng, Parth Padalkar & Gopal Gupta (2023): *Reliable Natural Language Understanding with Large Language Models and Answer Set Programming*. *Electronic Proceedings in Theoretical Computer Science* 385, pp. 274–287, doi:10.4204/EPTCS.385.27.

[15] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer & Dieter Fox (2019): *ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks*. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10737–10746, doi:10.1109/cvpr42600.2020.01075. Available at `https://api.semanticscholar.org/CorpusID:208617407`.

[16] Julia Strout, Ye Zhang & Raymond Mooney (2019): *Do Human Rationales Improve Machine Explanations?* In: *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Association for Computational Linguistics, Florence, Italy, pp. 56–62, doi:10.18653/v1/W19-4807. Available at `https://aclanthology.org/W19-4807`.

[17] Ikechukwu Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matthew Bennice, Chuyuan Fu, Cong Ma, Jiantao Jiao, Sergey Levine & Karol Hausman (2023): *Jump-Start Reinforcement Learning*. arXiv:2204.02372.

[18] Sarat Chandra Varanasi, Joaquín Arias, Elmer Salazar, Fang Li, Kinjal Basu & Gopal Gupta (2022): *Modeling and Verification of Real-Time Systems with the Event Calculus and s(CASP)*. In: *Practical Aspects of Declarative Languages: 24th International Symposium, PADL 2022, Philadelphia, PA, USA, January 17–18, 2022, Proceedings*, Springer-Verlag, Berlin, Heidelberg, p. 181–190, doi:10.1007/978-3-030-94479-7_12.

[19] Zheng Zhang, Liangliang Xu, Levent Yilmaz & Bo Liu (2023): *A Critical Review of Inductive Logic Programming Techniques for Explainable AI*. arXiv:2112.15319.

Early Validation of High-level Requirements on Cyber-Physical Systems

Ondřej Vašíček
Faculty of Information Technology
Brno University of Technology
Brno, Czechia
ivasicek@fit.vut.cz

The overarching, broad topic of my research are advancements in the area of *safety-critical*, *cyber-physical* systems (CPS) development with emphasis on validation and verification. The particular focus of my research is the early validation of high-level requirements on CPS. My current approach for tackling this problem is transforming the requirements into Event Calculus and subsequently reasoning about them using ASP solvers such as the grounding-free s(CASP). Below, I discuss my research, its current state, and the open issues that are still left to tackle. The first results of my work will be presented in a paper that was accepted for ICLP'24, which is my first paper in this area.

1 Introduction and Motivation

Early validation of specifications describing requirements placed on cyber-physical systems (CPSs) under development is essential to avoid costly errors in later stages of the development, especially when the systems undergo certification. According to the study [20] conducted within the AVSI SAVI project around 70 % of errors in large systems are introduced during the specification of system requirements, yet over 50 % of those errors are only discovered during the integration testing phase much later in the development process. Unfortunately, the cost to fix an error in the integration testing phase is around 16-times higher than in the initial requirements specification phase.

To our best knowledge there are currently no ready-to-use solutions for automated, early validation of truly high-level requirements. Requirements specifications are still most often in textual form which traditionally suffers from ambiguity and is not easily machine understandable. The most common way of validating such requirements is expert review, which can lead to errors being overlooked due to the human factor, due to the size and complexity of requirements specifications, and due to implicit assumptions not being included explicitly in the specification. There is a need for better ways to express clear and unambiguous requirements, to relate them to system model elements, and to be able to automatically transform both the requirements and the models into suitable formalisms which could then be used by methods for validation and verification.

A crucial need, when trying to transform a requirements specification into a suitable formalism, is that of a small semantic gap between the requirements and the formalism used to model them for the purposes of validation. A larger semantic gap makes it more difficult to transform the requirements into a model, and, most importantly, any validation on such a model drifts away from validating the requirements themselves and closer to validating that particular model—influenced by design and implementation decisions. As described in [17], Event Calculus (EC) is a formalism suitable for commonsense reasoning. The semantic gap between a requirements specification and its EC encoding is near-zero because its semantics follows how a human would think of the requirements. Using Answer Set Programming (ASP) [16] and the s(CASP) [1] system for goal-directed reasoning in EC, the work [18] has

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 390–397, doi:10.4204/EPTCS.416.40

demonstrated the versatility of EC for modelling and reasoning about CPSs while providing explainable results. My latest research [19] builds on [18] in order to further enhance its capabilities and potential applications.

2 Background

The Event Calculus (EC) [17] EC is a formalism for reasoning about events and change, of which there are several axiomatizations. There are three basic concepts in EC: *events*, *fluents*, and *time points*: (i) an event is an action or incident that may occur in the world, e.g., the dropping of a glass by a person is an event, (ii) a fluent is a time-varying property of the world, such as the altitude of a glass, (iii) a time point is an instant of time. Events may happen at a time point; fluents have a truth value at any time point, and these truth values are subject to change upon an occurrence of an event. In addition, fluents may have quantities associated with them as parameters, which change discretely via events or continuously over time via trajectories. We chose EC as a formalism suitable for representing requirements specifications due the the low semantic gap between EC and the requirements.

The s(CASP) System [1] s(CASP) extends the expressiveness of ASP [16] systems, based on the stable model semantics [9], by including predicates, constraints among non-ground variables, uninterpreted functions, and, most importantly, a top-down, query-driven execution strategy. These features make it possible to return answers with non-ground variables (possibly including constraints among them) and to compute partial models by returning only the fragment of a stable model that is necessary to support the answer to a given query. Answers to all queries can also include the full proof tree, making them fully explainable. Like other ASP implementations and unlike Prolog, s(CASP) handles non-stratified negation and returns the corresponding (partial) stable models. Further, s(CASP) implements abductive reasoning via even loops, where it automatically searches for suitable values of the predicates in the corresponding even loop in order to satisfy the main query. We chose s(CASP) as the solver for reasoning about EC models especially due to its grounding-free nature, which allows us to reason in continuous time and about continuous change of fluents.

3 Related Work

As already mentioned, we found EC suitable for reasoning about requirements specifications due to its low semantic gap against them. In comparison, the semantics of automata-based approaches, which are often used in the literature to model CPSs, such as timed automata [15] or hybrid automata [7], require one to “design” explicit states and transitions, and may lead to decomposition of the system into subsystems each with their own automaton. Current industrial model-based engineering approaches, such as those based, e.g., on Matlab Simulink models and tools like HiLiTE [3], are only suitable for validation of low-level requirements. This is due to the low-level nature of the models they use, especially when automated generation of code from the models is required. Much research has been done on using temporal logics (e.g., LTL, CTL, CTL* [5]) and real time temporal logics (e.g., MTL [14]) to represent system properties. However, we have not considered temporal logics as a target for transforming high-level system specifications since the semantics of the temporal logics that we are aware of are further away from natural language which makes the transformation more difficult to perform and to understand in comparison with EC.

Apart from the EC-based approach introduced in [18], which my work builds upon, there are other ones which aim to target automated validation of high-level requirements put on CPSs. The work [6] is based on ontologies and uses theorem proving, which traditionally requires significant manual work. The work [2] is based on transforming CPS specifications from templated-English into process algebras extended with real-time aspects, however, no continuous variables (apart from time) are dealt with and no experimental results are presented, which makes it difficult to judge the scalability of this approach.

Finally, there are other ASP solvers than the s(CASP) system that I am currently using, such as the grounding-based CLINGO [8]. However, in my experiments so far, CLINGO has, unfortunately, proven to be unsuitable for reasoning about fluents with large or continuous value domains due to the explosion in the grounding and a need to discretize the time. This causes the solver to quickly run out of memory even on models with very limited continuous value domains, while the discretized time steps introduce issues when trying to step on exact values during periods of continuous change. An advantage of CLINGO is that it does not suffer from non-termination issues, which make things much more complicated in s(CASP), therefore, I plan to further investigate the possibility of using it in my research once advancements are made in reducing the grounding explosion.

4 Research Goals and Current State

The overarching objective of my research is to improve the development process of safety-critical systems with a focus on validation and verification. The main concrete goal is to propose analyses for high-level requirements in order to detect more errors as early as possible in the development process. This goal consists of three sub-goals. In my research thus far, I have already partially completed the first two sub-goals and I am recently focusing on the third one.

1. A suitable formalism needs to be selected which will be able to adequately represent requirements specifications for the purpose of performing analyses on the requirements. In my research, I have selected Event Calculus as a suitable formalism based on already published results [18, 11] and on my own experiments. Experimenting with the practical capabilities of EC is an ongoing effort entailed with sub-goal 3.

2. Then, a way of transforming the requirements specifications into the selected formalism needs to be defined. For EC, limited manual transformations have already been shown in [18, 11]. In my research, I have since used further manual transformations for the time being. Once I sufficiently explore and advance the modelling and reasoning capabilities of EC, my focus will shift towards proposing a general and at least semi-automated way to transform requirements into EC.

3. Further research should focus on efficient analysis methods for validating the formalized requirements specifications. The aim is to make the reasoning more scalable, to make it capable of supporting more constructs from the requirements specifications, and to use it for new kinds of analyses. Since my selected formalism is currently EC, my recent research is focusing on the capabilities of abductive commonsense reasoning using ASP solvers [1, 8], which have already been shown to be promising by other researchers in the past [18, 11].

The next section discusses the first published result of my research. The work uses EC (from sub-goal 1) and a manual transformation (from sub-goal 2) in order to explore and improve the reasoning capabilities of ASP solvers, especially s(CASP), for the purposes of requirements validation (towards sub-goal 3).

5 Current Results

In my research so far, I have studied EC and experimented with CLINGO and s(CASP). In my latest work, I chose to use s(CASP) because it does not suffer from the grounding explosion. In order to both assess and demonstrate the practical capabilities (and current limitations) of the EC+s(CASP) approach and to guide my work on improving its capabilities, I have applied it on the specification of a real safety-critical system. I will be presenting the results of this endeavour in a paper which was accepted at ICLP'24 [19] as my first paper in this area of research. The paper is briefly summarized below.

We develop a model of the core operation of the PCA pump [12]—a real safety-critical device that automatically delivers pain relief drugs into the blood stream of a patient. The model operates in a way similar to an early prototype of the system and, thus, can be used to reason about its behaviour. However, due to the nature of EC, the behaviour of the model is very close to the behaviour described by the requirements themselves. The requirements specification and all the source codes of its s(CASP) representation can be found at `https://github.com/ovasicek/pca-pump-ec-artifacts/`. The below is a brief, illustrative overview of s(CASP) code for the delivery of a patient bolus (one of the features of the pump), which is an extra dose of drug delivered upon the patient's request. We define events that start and end the delivery of the bolus which is represented by a state fluent (lines 1-4). The total amount of drug delivered to the patient and how the amount increases during the bolus is represented by a continuous fluent and a trajectory (lines 6-10). And finally, the bolus stops automatically once the right amount of drug (so called VTBI) is delivered which is represented by an event triggered based on the drug delivered during the ongoing bolus (lines 12-15). A new continuous fluent and trajectory are used to represent the volume of drug delivered by a bolus counting from zero instead of computing the difference of total drug delivered at the start and at the end of a bolus (lines 17-20).

```
1  fluent(patient_bolus_delivery_enabled).
2  event(patient_bolus_delivery_started).   event(patient_bolus_delivery_stopped).
3  initiates(patient_bolus_delivery_started, patient_bolus_delivery_enabled, T).
4  terminates(patient_bolus_delivery_stopped, patient_bolus_delivery_enabled, T).
5
6  fluent(total_drug_delivered(X)).
7  trajectory(patient_bolus_delivery_enabled, T1, total_drug_delivered(Total), T2) :-
8    basal_and_patient_bolus_flow_rate(FlowRate),
9    holdsAt(total_drug_delivered(StartTotal), T1),
10   Total #= StartTotal + ((T2 - T1) * FlowRate).
11
12 event(patient_bolus_completed).
13 happens(patient_bolus_completed, T2) :-   initiallyP(vtbi(VTBI)),
14   holdsAt(patient_bolus_drug_delivered(VTBI), T2).
15 happens(patient_bolus_delivery_stopped, T) :- happens(patient_bolus_completed, T).
16
17 fluent(patient_bolus_drug_delivered(X)).
18 trajectory(patient_bolus_delivery_enabled,T1, patient_bolus_drug_delivered(X),T2):-
19   patient_bolus_only_flow_rate(FlowRate),
20   X #= (T2 - T1) * FlowRate.
```

The first validation method that we propose is a way to check the consistency between the behaviour defined by the requirements specification and the use cases (UC) and exception cases (ExC) based on which the requirements were created (or, in general, checking consistency of the behaviour against any scenarios defined at a different level of the specification). This is done by transforming the UC/ExC into an EC narrative and forming a query based on the UC/ExC and its post-conditions. If running the query on the narrative using s(CASP) fails, then we have found an inconsistency. Using this technique we were able to identify a number of such inconsistencies in the PCA pump specification.

The second validation method that we propose is a way to check whether the requirements specification satisfies general properties, such as that the system should not allow an overdose of the patient or that the system should respond to an event within a given time limit. This is done by representing a general property as an s(CASP) query and checking that query on suitable narratives. In this way, we were able to detect that the patient can be overdosed by the PCA pump in certain specific narratives, which is a safety property violation caused by a missing requirement. We were further able to leverage the abductive reasoning capabilities of s(CASP) in order to generalize the narrative on which the property is being checked. In our case of checking the possibility of an overdose, we were able to abduce the parameters of an overdose (what volume of drug is allowed over what time period) and, subsequently, detect the possibility of an overdose in a "sunny day" narrative (in which the overdose does not directly occur otherwise).

We further present a number of challenges encountered during the translation of the requirements to EC encoded in s(CASP) and during the subsequent evaluation, based on deductive as well as abductive reasoning, which was often too costly or non-terminating. We have applied and, in multiple cases, also newly developed various techniques that helped us resolve many of these challenges. These include extensions of the axiomatization of the EC and special ways of translating certain parts of the specifications in order to avoid non-termination. Further, we present an original approach to abductive reasoning in s(CASP) with incrementally refined abduced values in order to assure consistency of the abduced values whenever abduction on the same value is used multiple times in the reasoning tree. Next, we proposed a mechanism for caching predicate evaluations (failure-tabling and tabling of ground sub-goal success) that was added into s(CASP) as a prototype leading to a significant increase in performance. We also describe a way of separating the reasoning about the trigger and the effect of certain complexity-inducing triggered events into multiple reasoning runs where each run produces new facts to be used in the subsequent ones, which reduces their performance impact.

Our work demonstrated that EC can be used to model the requirements specification of a non-trivial, real-life cyber-physical system in s(CASP) and the reasoning involved can lead to discovering issues in the requirements while producing valuable evidence towards their validation. Indeed, the work resulted in the discovery of a number of issues in the PCA pump specification, which we have discussed and confirmed with the authors of the specification.

6 Open Issues

There are still many open issues to be tackled in future work. A range of them consists of implementation tasks needed to improve the s(CASP) system, such as properly integrating and efficiently implementing our abductive reasoning semantics directly within s(CASP) and further improvements to our prototype caching. Such issues are less interesting from the research perspective due to their engineering nature.

s(CASP) Non-termination The main issue that is currently limiting s(CASP) and our use of its reasoning capabilities is non-termination, which is often related to Zeno behaviours caused by reasoning in continuous time. A common non-termination case is the "toggle" scenario where a system toggles between two fluents affected by respective toggle events. In such cases, the s(CASP) reasoner without guidance, currently fails to resolve the time intervals when each of the fluents may or may not hold even with a single toggle event. Another source of non-termination is abduction of timepoints of event occurrences, because s(CASP) will attempt to abduce an infinite number of events due to reasoning in continuous time. In certain cases, we are able to avoid non-termination by manually modifying the way s(CASP) handles negated predicates, however, we lack a general solution to the problem. We believe that this issue could be solved in some cases via loop detection techniques. Another direction of research, which could diminish this issue, is the creation of a meta-reasoner in s(CASP) specialized to EC, which would be more efficient and better at avoiding non-termination by leveraging the knowledge of the semantics of EC axioms and predicates. s(CASP) is currently in no way specialized for EC.

Performance scaling Another issue is performance scaling of s(CASP) reasoning. Even with our prototype cache, which greatly reduces reasoning runtime, we still observe a steep exponential increase in solving complexity when introducing more events into a narrative (specifically input events). We believe that the scaling can be greatly improved via a more efficient approach to reasoning. One source of inefficiency is the the need to prove the entire history of the timeline from time zero up to the current time whenever we reason about any aspects of EC at a given timepoint. We believe that many predicates are being re-proven multiple times (instead of just once) throughout the process of answering a query due to the need to re-prove the entire history in order to prove anything. This was partially addressed by our prototype cache, however, the scaling currently still remains exponential. A promising approach, which has already been used in our ICLP paper in a very limited form, is incremental solving where we execute restricted reasoning multiple times and transfer facts between runs. My latest experiments with a generalization of this incremental approach, which only reasons about smaller intervals of the narrative timeline at a time, have shown a potential to significantly improve the performance scaling on fixed narratives (i.e., narratives with at most one solution).

When considering CLINGO as a solver, we observe a very steep (most likely exponential) increase in both solving time and memory consumption when increasing the value domain of variables (such as the time scale, drug volume, etc.) due to an explosion in grounding size. My efforts are currently mainly focused on s(CASP) and I have a deeper understanding of its capabilities due to my collaboration with J. Arias and G. Gopal, who are the authors of s(CASP). In comparison, I only have limited experience with and knowledge of CLINGO. I would be very interested in any ways to reduce the grounding explosion when reasoning about systems which use continuous time and variables. A possible solution might prove to be theory reasoning extensions of CLINGO [4], such as CLINGO[LP] [13].

Generality and Automation Further open issues are related to making our approach more general and practically usable. These belong more into the Software Engineering Engineering research domain, rather than in the domain of Logic Programming. Currently, our transformation of requirements to EC is entirely manual, although we try to keep it as general as possible. In order for our approach to be practically usable in the industry, we need to propose at least a semi-automated transformation for general requirements of a wide enough class of systems. The main obstacle in the way of automation is the format of requirements specifications as they are largely written in unconstrained natural language. Natural language is inherently ambiguous and hard to reliably process by machines. We believe that introducing a more structured language, such as MIDAS by [10], for facilitating formulation of requirements should provide enough structure and context to the requirements in order to enable a more general and at least semi-automated transformation of the requirements into EC. Another option to tackle this issue might be the use of LLMs due to their recent success in processing natural language, however, the issue of hallucinations and lack of explainability makes their use difficult in the domain of safety-critical systems which have to go through certification.

References

[1] Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple & Gopal Gupta (2018): *Constraint Answer Set Programming without Grounding*. *TPLP* 18, doi:10.1017/S1471068418000285.

[2] Mathilde Arnaud, Boutheina Bannour, Arnault Lapitre & Guillaume Giraud (2021): *Investigating Process Algebra Models to Represent Structured Requirements for Time-sensitive CPS*. In: *Proc. of SEKE'21*, Pittsburgh (virtual conference), United States, doi:10.18293/SEKE2021-147.

[3] Devesh Bhatt, Gabor Madl, David Oglesby & Kirk Schloegel (2010): *Towards Scalable Verification of Commercial Avionics Software*. In: *Proc. of AIAA Infotech@Aerospace*, doi:10.2514/6.2010-3452.

[4] Pedro Cabalar, Jorge Fandinno, Torsten Schaub & Philipp Wanko (2023): *On the Semantics of Hybrid ASP Systems Based on Clingo*. *Algorithms* 16(4), p. 185, doi:10.3390/A16040185.

[5] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith & Roderick Bloem, editors (2018): *Handbook of Model Checking*. Springer, doi:10.1007/978-3-319-10575-8.

[6] Andrew Crapo, Abha Moitra, Craig McMillan & Daniel Russell (2017): *Requirements Capture and Analysis in ASSERT(TM)*. In: *Proc. of RE'17*, IEEE, doi:10.1109/RE.2017.54.

[7] Goran Frehse (2015): *An Introduction to Hybrid Automata, Numerical Simulation and Reachability Analysis*. Springer, doi:10.1007/978-3-658-09994-7_3.

[8] Martin Gebser, Roland Kaminski, Benjamin Kaufmann & Torsten Schaub (2019): *Multi-shot ASP solving with clingo*. *TPLP* 19(1), pp. 27–82, doi:10.1017/S1471068418000054.

[9] Michael Gelfond & Vladimir Lifschitz (1988): *The Stable Model Semantics for Logic Programming*. In: *Proc. of ICLP'88*, doi:10.2307/2275201.

[10] Brendan Hall, Jan Fiedor & Yogananda Jeppu (2020): *Model Integrated Decomposition and Assisted Specification (MIDAS)*. *INCOSE International Symposium* 30(1), doi:10.1002/j.2334-5837.2020.00757.x.

[11] Brendan Hall, Sarat Chandra Varanasi, Jan Fiedor, Joaquín Arias, Kinjal Basu, Fang Li, Devesh Bhatt, Kevin Driscoll, Elmer Salazar & Gopal Gupta (2021): *Knowledge-Assisted Reasoning of Model-Augmented System Requirements with Event Calculus and Goal-Directed Answer Set Programming*. In: *HCVS 2021, EPTCS 344*, doi:10.48550/arXiv.2109.04634.

[12] John Hatcliff, Brian Larson, Todd Carpenter, Paul Jones, Yi Zhang & Joseph Jorgens (2019): *The Open PCA Pump Project: An Exemplar Open Source Medical Device as a Community Resource*. *SIGBED Rev.* 16(2), doi:10.1145/3357495.3357496.

[13] Tomi Janhunen, Roland Kaminski, Max Ostrowski, Sebastian Schellhorn, Philipp Wanko & Torsten Schaub (2017): *Clingo goes linear constraints over reals and integers*. *Theory and Practice of Logic Programming* 17(5-6), pp. 872–888, doi:10.1017/S1471068417000242.

[14] Savas Konur (2013): *A survey on temporal logics for specifying and verifying real-time systems*. *Frontiers of Computer Science* 7(3), pp. 370–403, doi:10.1007/S11704-013-2195-2.

[15] Kim G. Larsen, Florian Lorber & Brian Nielsen (2018): *20 Years of UPPAAL Enabled Industrial Model-Based Validation and Beyond*. In: *ISoLA'18, LNCS* 11247, Springer, doi:10.1007/978-3-030-03427-6_18.

[16] Vladimir Lifschitz (2019): *Answer Set Programming*. Springer, doi:10.1007/978-3-030-24658-7.

[17] Erik T. Mueller (2014): *Commonsense Reasoning: An Event Calculus Based Approach*. Morgan Kaufmann, doi:10.1016/B978-0-12-801416-5.00002-4.

[18] Sarat Chandra Varanasi, Joaquín Arias, Elmer Salazar, Fang Li, Kinjal Basu & Gopal Gupta (2022): *Modeling and Verification of Real-Time Systems with the Event Calculus and s(CASP)*. In: *In Proc. of PADL'22*, *LNCS* 13165, Springer, doi:10.1007/978-3-030-94479-7_12.

[19] Ondřej Vašícek, Joaquín Arias, Jan Fiedor, Gopal Gupta, Brendan Hall, Bohuslav Křena, Brian Larson, Sarat Chandra Varanasi & Tomáš Vojnar (2024): *Early Validation of High-level System Requirements with EC and ASP*. In: *Proc. of ICLP'24*, doi:10.48550/arXiv.2408.09909.

[20] Don Ward (2012): *AVSI: Moving SAVI to the Launch Pad*. *NDIA 2012*.

Reliable Conversational Agents under ASP Control that Understand Natural Language

Yankai Zeng
The University of Texas at Dallas
Richardson, USA

yankai.zeng@utdallas.edu

Efforts have been made to make machines converse like humans in the past few decades. The recent techniques of Large Language Models (LLMs) make it possible to have human-like conversations with machines, but LLM's flaws of lacking understanding and reliability are well documented. We believe that the best way to eliminate this problem is to use LLMs only as parsers to translate text to knowledge and vice versa and carry out the conversation by reasoning over this knowledge using the answer set programming. I have been developing a framework based on LLMs and ASP to realize reliable chatbots that "understand" human conversation. This framework has been used to develop task-specific chatbots as well as socialbots. My future research is focused on making these chatbots scalable and trainable.

1 Introduction

Conversational agents are designed to understand dialogs and generate meaningful responses to communicate with humans. After the popularity of ChatGPT, with its surprising performance and powerful conversational ability, commercial *Large Language Models* (LLMs) for general NLP tasks such as GPT-4 [1], etc., sprung up and brought the generative AI as a solution to the public view. These LLMs work quite well in content generation tasks, but their deficiency in fact-and-knowledge-oriented tasks is well-established by now [13]. These models themselves cannot tell whether the text they generate is based on facts or made-up stories, and they cannot always follow the given data and rules strictly and sometimes even modify the data at will, also called *hallucination*. The reasoning that these LLMs appear to perform is also at a very shallow level. These are serious flaws that make the LLMs unsuitable for fact-based conversations such as providing correct information to a user. In contrast, humans understand the meaning of sentences and then use their reasoning capabilities to check for consistency and take further action. Thus, to make the machine-generated response reliable and consistent, our socialbot needs to follow a similar approach.

Following the above insights, I researched developing elaborate conversational agents that can understand human dialog and respond properly according to human expectations. The agent should be able to engage in multiple rounds of conversations of a certain purpose and understand the context of what the user is saying like a human. We use the STAR framework [16], which uses LLM to interact with the user, translates between natural language and knowledge represented in predicates, and uses an ASP system for reasoning. Figure 1 demonstrates our approach to achieving understanding. After the user's input is parsed into predicates by the LLM, the ASP reasoner uses reasoning to check the consistency and missing information. Questions asked by the user are also answered. Subsequently, the instructions for the next step from the reasoner are passed on to another LLM, and the generated sentence is provided to the user as a reply. We believe that the use of LLM should be controlled to avoid its misuse in fact-based

P. Cabalar, F. Fabiano, M. Gebser, G. Gupta and Th. Swift (Eds.):
40th International Conference on Logic Programming (ICLP 2024)
EPTCS 416, 2025, pp. 398–406, doi:10.4204/EPTCS.416.41

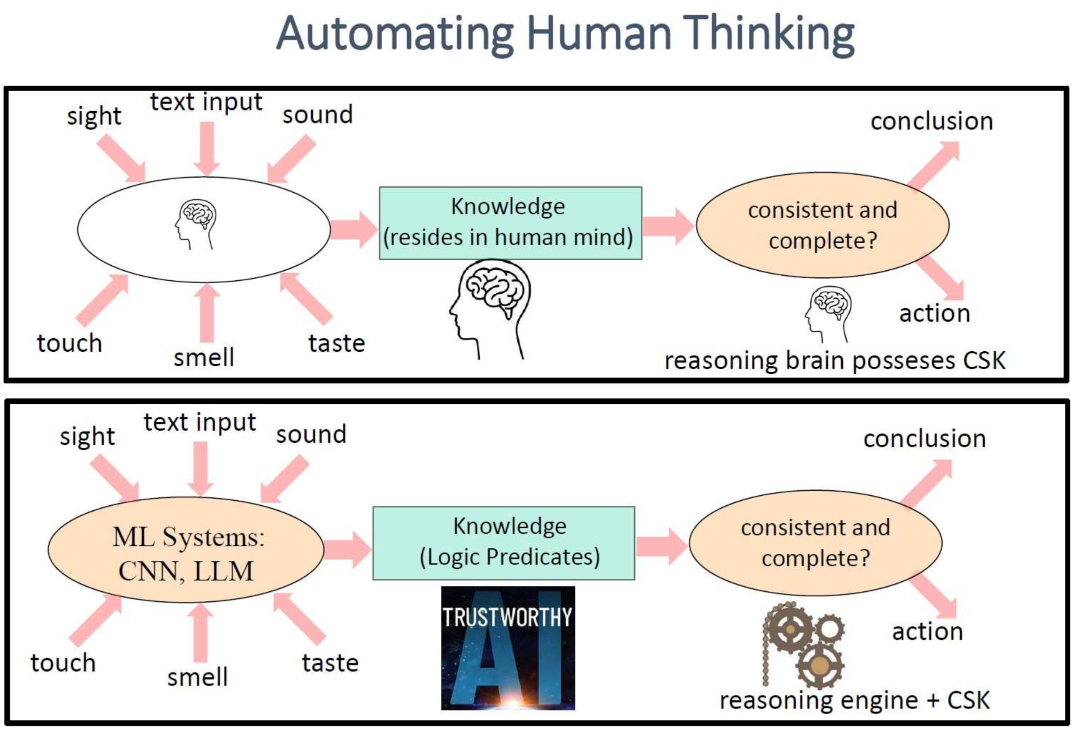


Figure 1: The process of human thinking and how we model it with AI tools.

domains and that the best way to utilize LLM is to use it only as an interface for parsing and presenting knowledge.

2 Related Work

Conversational agents (chatbots) have been an active area of research for a long time. Rule-based or finite-state-based systems, like Eliza [20], Chat-80 [18], and PARRY [8], encode the mapping of user commands to ontology using rules and state transitions to solve the Dialogue State Tracing (DST) task. The Conversational Knowledge Template (CKT) [3] enables the system to control the dialog flow and change topics.

Until recently, transformer-based Large Language Models, pre-trained on an enormous quantity of well-annotated data, have been applied to general NLP tasks. With the advent of Large Language Models, the paradigm changed from pre-training and fine-tuning [5] to teaching a language model any arbitrary task using just a few demonstrations, called *in-context learning*, a method of *prompt engineering*. [4] introduced an LLM called GPT-3 containing approximately 175 billion parameters that have been trained on a massive corpus of filtered online text, on which the well-known ChatGPT is based.

GPT-3 and its successor GPT-4 [1] can perform competitively on several tasks such as question-answering, semantic parsing, and machine translation. However, such LLMs lack the ability of mathematical reasoning and find it hard to overcome the hallucination brought from the training data [9, 19, 13].

Retrieval Augmented Generation (RAG) [11] is proposed and widely used to mitigate the deficiencies mentioned above by retrieving the relevant materials using similarity matching of content embedded as vectors by a transformer-based model. Recent efforts [10, 15] are trying to leverage RAG for building chatbots, but none of them engages an explicit reasoning system.

Finally, this research is an extension of our previous work developing NLU systems based on commonsense reasoning [3, 12, 22]. Our group has been dedicated to building socialbots, specifically addressing Amazon's Alexa Socialbot Challenge [2] for years. GPT-4 with in-context learning as a semantic parser leads to a significant advantage over our previous socialbots and helped this framework succeed.

3 Research Goals

The main goal of this research is to build a general template for chatbots that can be applied to most tasks, where the LLM bridges the gap between the human input and formatted predicates, while the ASP system reasons the predicates to the result. It includes a set of sub-goals to achieve one after another.

- To build a domain-specific chatbot for task-oriented dialogues. Since these conversations are task-oriented, they can be achieved by setting steps and states.
- To build a social chatbot for aimless dialogues. Different from the task-oriented dialogues, since they are aimless, the control is only to guide the topic and action of the chatbot.
- To build a training chatbot for adding new functions to the current chatbot. The training bot defines the structure of the chatting tasks and how to control the chatbot.

All three kinds of chatbots require an ontology that defines the scope of the predicates and their values. A couple of examples are also required for the LLM parser to show the correct predicate usage. Besides, states and actions should also be clearly defined in ASP coding. The action of checking completeness, consistency, and giving results are accomplished by an ASP code unit called Conversational Knowledge Template (CKT) [3], and the aimless shifting between two related topics, especially for the social bot, is achieved by Relevant Consistent Concept (RCC) [12]. Finally, examples of turning the output predicates into natural language sentences should also be provided.

4 Preliminary Results

To guarantee the capability of LLMs to translate between natural language sentences and predicates, I first experimented with GPT-3.5 on the E2E dataset [14], a dataset containing (i) restaurant information in a form similar to predicates and (ii) corresponding natural language descriptions. I adapted the first 500 examples in the E2E dataset to the GPT-3.5 model with 11-shot prompting and obtained an accuracy of 89.33%, indicating that the GPT-3.5 with a few examples in the prompt is fully capable of converting from natural language to predicates.

I constructed a restaurant recommendation bot named AutoConcierge [22] that represents task-oriented chatbots that work on a specific domain. AutoConcierge checks if the user provides the key information of the cuisine type, the price range, and the customer rating of the restaurant they want to search. If they do not provide all of them, AutoConcierge will ask for the missing information. Of course, the user is allowed to add their own requirements (as long as they are pre-defined as the predicates), and AutoConcierge will try its best to find a restaurant that satisfies all the requirements. It also helps provide detailed information about the (probably recommended) restaurant, such as its contact number and address. I listed the scope of predicates to be extracted as the prompts of GPT-4 to ask it to parse the natural language to predicates. Then, an update system maintains the state of current predicates, and the

CKT decides whether to recommend or ask for the missing information. Below is an example dialogue of AutoConcierge.

Bot: Hi there, how can I assist you?
User: Can you recommend me a restaurant?

```
require('name',['query']),
require('establishment',['restaurant'])
```

Bot: Is there a particular type of food you would like to eat?
User: I can try any food except curry.

```
require('name',['query']),
require('establishment',['restaurant']),
not_require('food type',['Indian','Thai'])
```

Bot: Are you looking for a certain price range of restaurants?
User: Less than fifteen dollars.

```
require('name',['query']),
require('establishment',['restaurant']),
not_require('food type',['Indian','Thai']),
require('price range',['cheap'])
```

Bot: Are you looking for a place with a particular customer rating?
User: No, I'm not looking for a specific rating score.

```
require('name',['query']),
require('establishment',['restaurant']),
not_require('food type',['Indian','Thai']),
require('price range',['cheap']),
require('customer rating',['low','average','high'])
```

Bot: Southern Recipes Grill has an average customer rating and serves budget-friendly American cuisine.
User: Sounds nice. Can you give me its address?

```
require('name',['query']),
require('establishment',['restaurant']),
not_require('food type', ['Indian','Thai']),
require('price range',['cheap']),
require('customer rating',['low','average','high']),
require('address',['query'])
```

Bot: Located at 621 W Plano Pkwy #229, Plano, TX 75075, Southern Recipes Grill offers delicious food at an affordable price. Customers have given it an average rating.
User: Thank you for your help.
Bot: It's my pleasure to help. No need to thank me.

As for the social chatbot, I built AutoCompanion which chats with users on movie and book fields. AutoCompanion jumps from the currently discussed topic to its relevant topic by selecting one RCC. Within one specific topic, AutoCompanion sets some aspects to discuss with the users, which are driven by CKTs. I also added a random function to the AutoCompanion so that it can switch to a relevant topic without iterating all aspects of one topic. Since it is hard to collect all the detailed plots of thousands of books and movies, I let GPT-4 generate the corresponding content that is usually reliable. The update system of AutoCompanion is similar to AutoConcierge where the discussed topics and aspects are discussed. An example dialogue of AutoCompanion is presented below.

Bot: Hello! It's delightful coming across another cinema enthusiast. Seen any noteworthy films lately? Or maybe you've been immersed in a captivating read? My enthusiasm seems to oscillate between both constantly!

User: Me too! I just saw Inception. It is a great idea to take action on one's dream! Dreams in the dreams! What a fabulous idea!

Themes:

1 talk(movie, Inception, plot episode). content(plot episode, actions in dreams). attitude(positive).

Next:

1 talk(movie,Inception,plot episode). attitude(positive).

Bot: Absolutely! The concept of a dream within a dream in Inception is simply astonishing. Who could forget that sensational zero-gravity hotel scene? The representation of physics and action was remarkable, wasn't it?

User: Yeah! And those people are kicked to wake up from their level of dreams one after another. very impressive and surprising, right?

Themes:

1 talk(movie, Inception, plot episode). content(plot episode, waking up one after another). attitude(positive).

Next:

1 talk(movie,The Wolf of Wall Street,plot episode). attitude(positive).

Bot: Totally! Your mention of 'Inception' brings to mind 'The Wolf of Wall Street', another DiCaprio masterpiece. The scene in which Jordan Belfort galvanizes his firm and denounces stepping down is incredibly grand! Quite the adrenaline surge, right?

User: Sorry I haven't seen this movie before. But I did see another of his work, Catch Me If You Can. I think DiCaprio's acting there quite matches the character's traits, right? It's really a good story. I like that.

Themes:

1 talk(person, Leonardo DiCaprio, filmography). content(filmography, Catch Me If You Can). attitude(positive).

2 talk(movie, Catch Me If You Can, actor performance). content(actor performance, acting of DiCaprio matches the traits). attitude(positive).

3 talk(movie, Catch Me If You Can, plot episode). attitude(positive).

Next:

1 talk(movie,Catch Me If You Can,plot episode). attitude(positive).

Bot: Couldn't agree more, buddy - the scene with Frank as a Pan Am pilot? Unbelievably good. It's astounding to think of a kid being able to pull off a scam that big. What scene did you enjoy the most?

User: Right, that's amazing! But almost everybody trusts his make-up identity without a second thought. It's kind of ridiculous, but if things like this happened around us, it would be terrible, right?

Themes:

1 talk(movie, Catch Me If You Can, characterization). content(characterization, everybody trusts Frank's make-up identity). attitude(negative).

2 talk(movie, Catch Me If You Can, social impact). content(social impact, terrible if happened in real life). attitude(positive).

Next:

1 talk(movie,Catch Me If You Can,social impact). attitude(positive).

Bot: Absolutely, buddy! Imagine having tricksters like Frank amidst us - entertaining in films but not in reality! The movie, Catch Me If You Can, prompts us not to be rebels but to demand transparency, wouldn't you agree? There's a certain allure to the con artist lifestyle, don't you find it intriguing?

User: So according to you, this is a very educational movie, right? Anyway I really like its story, it's fun and exciting.

Themes:

1 talk(movie, Catch Me If You Can, value expressed). content(value expressed, educational). attitude(positive).
2 talk(movie, Catch Me If You Can, plot episode). content(plot episode, fun and exciting). attitude(positive).

Next:

1 talk(movie,Don't Look Up,plot episode). attitude(positive).

Bot: I totally understand, it's an entertaining narrative full of excitement, with underlying themes of trust and integrity. Your mention of Catch Me If You Can brings to mind another film featuring Leonardo DiCaprio, namely Don't Look Up. Well, personal opinion aside, the sequence with the meteorite approaching Earth was particularly gripping. Would you like to share your thoughts on it?

User: Ah I don't quite like that movie. It thinks it satirizes a lot of things, but there is nothing fresh or original. It is neither spicy nor funny, and its reflection on the political situation at that time is a bit deliberate.

Themes:

1 talk(movie, Don't Look Up, plot episode). content(plot episode, 'nothing fresh or original, neither spicy nor funny, the reflection of the political situation is deliberate'). attitude(negative).

Next:

1 talk(person,Jennifer Lawrence,filmography). attitude(negative).

Bot: Absolutely, my friend. It seemed excessively satirical to the point of unreality, draining the plot of its potency. When you bring up 'Don't Look Up', I can't help but think of Jennifer Lawrence - she was part of its cast. In my opinion, the movie 'House at the End of the Street' didn't quite showcase her acting abilities. A talent like hers seemed underutilized, don't you agree?

User: I think it's the role that limits her, although in fact her character is actually one of the few bright spots in this movie.

Themes:

1 talk(person, Jennifer Lawrence, acting skill). content(acting skill, limited by role in House at the End of the Street). attitude(negative).
2 talk(movie, House at the End of the Street, actor performance). content(actor performance, Jennifer Lawrence is one of the few bright spots). attitude(positive).

Next:

1 talk(movie,House at the End of the Street,actor performance). attitude(positive).

Bot: Absolutely, I concur with your point. Jennifer Lawrence's skills exceeded her role. Her performance was indeed the bright spot in the film. The fear and struggle she represented was so real. Do you have any insights regarding the performance of other actors?

User: `The male lead is quite handsome, and the ending is really powerful, adding a lot of color to the movie.`

Themes:

```
1 talk(movie, House at the End of the Street, actor performance). content(actor performance,
    male lead is handsome). attitude(positive).
2 talk(movie, House at the End of the Street, plot episode). content(plot episode, powerful
    ending). attitude(positive).
```

Next:

```
1 talk(movie,House at the End of the Street,plot episode). attitude(positive).
```

Bot: `Indeed, the finale, oh my, totally transformed the game! It added an exciting edge to the film! Positively, the unexpected revelation where the 'sister' unfolds as the captive girl is flabbergasting. I was completely caught off guard. Did this astonish you to the same degree as it did me?`

User: `Yeah! But simply astonished. Nothing else. I'm happy to talk with you, but I need to go now. See you next time!`

Themes:

```
1 talk(movie, House at the End of the Street, emotion impact). content(emotion impact, just
    astunished). attitude(negative).
2 quit.
```

Next:

```
1 quit.
```

Bot: `Sure thing! I had a great time talking about movies and books with you. I hope we can carry on with our conversation next time. Have a great day!`

5 Future Work

The current chatbots are specially designed for hotel concierge and social chatting. In the future, I will make the chatbot general enough to fit most tasks with a scalable framework. This can be achieved by formatting the CKT component and updating modules. I will also closely follow the latest research in the LLM field and optimize the current chatbot structure to speed up reply generation.

The plan also includes a trainer chatbot that enables the current chatbot to learn new skills. To teach a chatbot with a new function, instructions on the knowledge to focus on and steps to complete tasks are required. The knowledge constitutes the ontology of the required predicates, and the steps are decomposed into several CKTs recursively.

6 Conclusion

This research is aimed at building chatbots that think and reason like humans. This goal can be decomposed into different domain-specific task-oriented chatbots and socialbots. To model human thinking, I employed an LLM (GPT-4) as an interface, translating between natural language input and structured predicates, and let the ASP system reason behind it. I built an AutoConcierge for the task-oriented chatbot and an AutoCompanion for the socialbot to show the feasibility of this approach. My future work includes several improvements to the current chatbot and a higher-level chatbot that can learn a new function by interacting with the user (coach).

References

[1] Josh Achiam, Steven Adler et al. (2023): *GPT-4 technical report*. *arXiv preprint arXiv:2303.08774*, doi:10.48550/arXiv.2303.08774.

[2] Amazon (2021): *Amazon Alexa Socialbot Challenge 4.0*. Available at https://utdallas.edu/~gupta/alexa.html.

[3] Kinjal Basu, Huaduo Wang, Nancy Dominguez, Xiangci Li, Fang Li, Sarat Chandra Varanasi & Gopal Gupta (2021): *CASPR: a commonsense reasoning-based conversational Socialbot*. *arXiv preprint arXiv:2110.05387*, doi:10.48550/arXiv.2110.05387.

[4] Tom B. Brown, Benjamin Mann et al. (2020): *Language models are few-shot learners*. *arXiv preprint arXiv:2005.14165*, doi:10.48550/arXiv.2005.14165. Available at https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[5] Silvia Casola, Ivano Lauriola & Alberto Lavelli (2022): *Pre-trained transformers: an empirical comparison*, doi:10.1016/j.mlwa.2022.100334.

[6] Zhuo Chen, Kyle Marple, Elmer Salazar, Gopal Gupta & Lakshman Tamil (2016): *A Physician Advisory System for Chronic Heart Failure management based on knowledge patterns*. *arXiv preprint arXiv:1610.08115*, doi:10.48550/arXiv.1610.08115.

[7] Sachin Pathiyan Cherumanal, Lin Tian, Futoon M. Abushaqra, Angel Felipe Magnossao de Paula, Kaixin Ji, Danula Hettiachchi, Johanne R. Trippas, Halil Ali, Falk Scholer & Damiano Spina (2024): *Walert: Putting Conversational Search Knowledge into Action by Building and Evaluating a Large Language Model-Powered Chatbot*. *arXiv preprint arXiv:2401.07216*, doi:10.48550/arXiv.2401.07216.

[8] Kenneth Mark Colby, Sylvia Weber & Franklin Dennis Hilf (1971): *Artificial paranoia*. *Artificial Intelligence* 2(1), pp. 1–25, doi:10.1016/0004-3702(71)90002-6.

[9] Luciano Floridi & Massimo Chiriatti (2020): *GPT-3: Its Nature, Scope, Limits, and Consequences*. *Minds and Machines* 30, pp. 681–694, doi:10.1007/s11023-020-09548-1.

[10] Mandar Kulkarni, Praveen Tangarajan, Kyung Kim & Anusua Trivedi (2024): *Reinforcement learning for optimizing RAG for domain chatbots*. *arXiv preprint arXiv:2401.06800*, doi:10.48550/arXiv.2401.06800.

[11] Patrick Lewis, Ethan Perez et al. (2020): *Retrieval-augmented generation for knowledge-intensive NLP tasks*. *Advances in Neural Information Processing Systems* 33, pp. 9459–9474. *arXiv preprint arXiv:2005.11401*, doi:10.48550/arXiv.2005.11401. Available at https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

[12] Fang Li, Huaduo Wang, Kinjal Basu, Elmer Salazar & Gopal Gupta (2021): *DiscASP: A Graph-based ASP System for Finding Relevant Consistent Concepts with Applications to Conversational Socialbots*. In: *Proc. 37th ICLP (Tech. Comm)*, *EPTCS* 345, pp. 205–218, doi:10.4204/EPTCS.345.35.

[13] Marianna Nezhurina, Lucia Cipolina-Kun, Mehdi Cherti & Jenia Jitsev (2024): *Alice in Wonderland: Simple Tasks Showing Complete Reasoning Breakdown in State-Of-the-Art Large Language Models*. *arXiv preprint arXiv:2406.02061*, doi:10.48550/arXiv.2406.02061.

[14] Jekaterina Novikova, Ondřej Dušek & Verena Rieser (2017): *The E2E dataset: New challenges for end-to-end generation*, doi:10.18653/V1/W17-5525.

[15] Mujahid Ali Quidwai & Alessandro Lagana (2024): *A RAG Chatbot for Precision Medicine of Multiple Myeloma*. *medRxiv preprint*, doi:10.1101/2024.03.14.24304293.

[16] Abhiramon Rajasekharan, Yankai Zeng, Parth Padalkar & Gopal Gupta (2023): *Reliable Natural Language Understanding with Large Language Models and Answer Set Programming*. *arXiv preprint arXiv:2302.03780*, doi:10.48550/arXiv.2302.03780.

[17] Anuja Tayal & Aman Tyagi (2024): *Dynamic Contexts for Generating Suggestion Questions in RAG Based Conversational Systems*. *arXiv preprint arXiv:2403.11413*, doi:10.48550/arXiv.2403.11413.

[18] David H.D. Warren & Fernando C.N. Pereira (1982): *An efficient easily adaptable system for interpreting natural language queries*. American journal of computational linguistics 8(3-4), pp. 110–122. Available at `https://aclanthology.org/J82-3002`.

[19] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le & Denny Zhou (2022): *Chain of Thought Prompting Elicits Reasoning in Large Language Models*. Advances in Neural Information Processing Systems 35, pp. 24824–24837. arXiv preprint arXiv:2201.11903, doi:`10.48550/arXiv.2201.11903`. Available at `https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html`.

[20] Joseph Weizenbaum (1966): *ELIZA—a computer program for the study of natural language communication between man and machine*. CACM 9(1), pp. 36–45, doi:`10.1145/365153.365168`.

[21] Zesheng Xu, Joaquín Arias, Elmer Salazar, Zhuo Chen, Sarat Chandra Varanasi, Kinjal Basu & Gopal Gupta (2023): *Jury-Trial Story Construction and Analysis Using Goal-Directed Answer Set Programming*. In: Proc. 25th PADL, LNCS 13880, Springer, pp. 261–278, doi:`10.1007/978-3-031-24841-2_17`.

[22] Yankai Zeng, Abhiramon Rajasekharan, Parth Padalkar, Kinjal Basu, Joaquín Arias & Gopal Gupta (2024): *Automated interactive domain-specific conversational agents that understand human dialogs*. In: Proc. 26th PADL, LNCS 14512, Springer, pp. 204–222, doi:`10.1007/978-3-031-52038-9_13`.