EPTCS 421

Proceedings of the

19th International Workshop on

Logical and Semantic Frameworks, with Applications

Goiânia, Brazil, 18th-20th September 2024

Edited by: Cynthia Kop and Helida Salles Santos

Published: 6th June 2025
DOI: 10.4204/EPTCS.421
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Table of Contents i

Preface ii
Cynthia Kop and Helida Salles Santos

Properties of UTxO Ledgers and Programs Implemented on Them 1
Polina Vinogradova and Alexey Sorokin

Query Answering in Lattice-based Description Logic 21
Krishna Manoorkar and Ruoding Wang

Fuzzy Lattice-based Description Logic 44
Yiwen Ding and Krishna Manoorkar

Regional, Lattice and Logical Representations of Neural Networks 64
Sandro Preto and Marcelo Finger

Nominal Equational Rewriting and Narrowing 80
Mauricio Ayala-Rincón, Maribel Fernández, Daniele Nantes-Sobrinho and Daniella Santaguida

Towards an Analysis of Proofs in Arithmetic 98
Alexander Leitsch, Anela Lolić and Stella Mahler

An Execution Model for RICE 112
Steven Libby

Paraconsistent Relations as a Variant of Kleene Algebras 130
Juliana Cunha, Alexandre Madeira and Luís S. Barbosa

Preface

This volume contains the post-proceedings of the 19th Workshop on Logical and Semantic Frameworks with Applications (LSFA 2024), which was held in Goiânia, the capital of Goiás state in Brazil, from September 18 to September 20, 2024.

The aim of the LSFA series of workshops is bringing together theoreticians and practitioners to promote new techniques and results, from the theoretical side, and feedback on the implementation and use of such techniques and results, from the practical side. LSFA includes areas such as proof and type theory, equational deduction and rewriting systems, automated reasoning and concurrency theory.

The 19th International Workshop on Logical and Semantic Frameworks, with Applications continues the successful workshops held in Natal (2006), Ouro Preto (2007), Salvador (2008), Brasilia (2009), Natal (2010), Belo Horizonte (2011), Rio de Janeiro and Niterói (2012), São Paulo (2013), Brasília (2014), Natal (2015), Porto (2016), Brasília (2017), Fortaleza (2018), Natal (2019), Bahia (2020), Buenos Aires (virtual) (2021), Belo Horizonte (2022), and Rome (2023).

For LSFA 2024, nine regular papers were accepted for presentation out of ten submissions, with three or four reviewers per submission. After the meeting, revised versions of the papers were reviewed again, from which eight regular papers were finally included in this volume. In addition, the workshop program included two talks by distinguished invited speakers Heloisa Camargo (Universidade Federal de São Carlos) and Maribel Fernandez (King's College London), and two tutorial sessions by distinguished invited lecturers Marcelo Finger (IME - USP) and Anderson Paiva Cruz (Instituto Metrópole Digital UFRN). We express our sincere gratitude to them.

We want to thank the PC members and the additional reviewers for doing a great job providing high-quality reviews. Many thanks also go to the LSFA 2024 organizers, Daniel Ventura, Bruno Silvestre, Thaynara Arielly de Lima and Wagner Sanz. All their valuable time was indispensable in guaranteeing the success of the workshop.

Cynthia Kop (LSFA 2024 PC chair)
Helida Santos Salles (LSFA 2024 PC co-chair)

Cynthia Kop and Helida Salles Santos (Eds); 19th Workshop on
Logical and Semantic Frameworks with Applications (LSFA'24)
EPTCS 421, 2025, pp. ii–iii, doi:10.4204/EPTCS.421.0

Program Committee

Vander Alves	Universidade de Brasília, Brazil
Luís Soares Barbosa	Universidade do Minho, Portugal
Benjamin Bedregal	Universidade Federal do Rio Grande do Norte, Brazil
Juliana Bowles	University of St Andrews, UK
Siddharth Bhaskar	James Madison University, USA
Frédéric Blanqui	INRIA, France
Humberto Bustince	Universidad Pública de Navarra, Spain
Jörg Endrullis	Vrije Universiteit Amsterdam, The Netherlands
Mário Florido	Universidade do Porto, Portugal
André Galdino	Universidade Federal de Catalão, Brazil
Alex Kavvos	University of Bristol, UK
Dohan Kim	University of Innsbruck, Austria
Cynthia Kop (chair)	Radboud University Nijmegen, The Netherlands
Thaynara Arielly de Lima	Universidade Federal de Goiás, Brazil
Mariano Moscato	AMA / NASA LaRC, USA
Flávio de Moura	Universidade de Brasília, Brazil
Cláudia Nalon	Universidade de Brasília, Brazil
Jorge Pérez	University of Groningen, The Netherlands
Renata Reiser	Universidade Federal de Pelotas, Brazil
Kristin Yvonne Rozier	Iowa State University
Thomas Rubiano	INRIA, France
Helida Santos (co-chair)	Universidade Federal do Rio Grande, Brazil
Regivan Santiago	Universidade Federal do Rio Grande do Norte, Brazil
José Solsona	Universidad ORT, Uruguay
Nora Szasz	Universidad ORT, Uruguay
Álvaro Tasistro	Universidad ORT, Uruguay
René Thiemann	University of Innsbruck, Austria
Deivid Vale	Radboud University Nijmegen, The Netherlands
Niccolò Veltri	Tallinn University of Technology, Estonia
Daniel Ventura	Universidade Federal de Goiás, Brazil
Niels van der Weide	Radboud University Nijmegen, The Netherlands

Organizing Committee

Daniel Ventura (local organizer)	INF, Universidade Federal de Goiás, Brazil
Thaynara Arielly de Lima	IME, Universidade Federal de Goiás, Brazil
Wagner Sanz	FAFIL, Universidade Federal de Goiás, Brazil
Bruno Silvestre	INF, Universidade Federal de Goiás, Brazil

External Reviewers

Fernando Bereta Dos Reis, João Barbosa, Jose Solsona, Manfred Schmidt-Schauß, Paulo Guilherme Santos

Properties of UTxO Ledgers and Programs Implemented on Them

Polina Vinogradova
Input Output Global
polina.vinogradova@iohk.io

Alexey Sorokin
Input Output Global
alex.sorokin@iohk.io

Trace-based properties are the gold standard for program behaviour analysis. One of the domains of application of this type of analysis is cryptocurrency ledgers, both for the purpose of analyzing the behaviour of the ledger itself, and any user-defined programs called by it, known as smart contracts. The (extended) UTxO ledger model is a kind of ledger model where all smart contract code is stateless, and additional work must be done to model stateful programs. We formalize the application of trace-based analysis to UTxO ledgers and contracts, expressing it in the languages of topology, as well as graph and category theory. To describe valid traces of UTxO ledger executions, and their relation to the behaviour of stateful programs implemented on the ledger, we define a category of simple graphs, infinite paths in which form an ultra-metric space. Maps in this category are arbitrary partial sieve-define homomorphisms of simple graphs. Programs implemented on the ledger correspond to non-expanding maps out of the graph of valid UTxO execution traces. We reason about safety properties in this framework, and prove properties of valid UTxO ledger traces.

Keywords: blockchain, ledger, smart contract, formal verification, specification, transition system, UTxO, properties, safety

1 Introduction

Cryptocurrency ledgers are distributed ledgers keeping records of digital currency. When a user submits a transaction to the network, each local record state is updated by applying the changes specified in the transaction. Cryptocurrency ledger behavior is often described by as a deterministic state transition systems [15, 7, 23, 31, 28]. The main functionality supported by blockchain ledgers programs is a single atomic operation: the application of a transaction to the given state, and all state updates can be decomposed into applications of single transactions. This makes small-step semantics well-suited to specify ledger behavior, as exemplified in both scientific research and in industrial implementations [30, 19].

The two most common ledger models are *account-based* (such as Ethereum [7]), and *UTxO-based*. A minimal account-based ledger records the state of a collection of accounts, and applying transactions to the ledger updates the account states, and transfers funds between accounts. In this work, we focus on the UTxO ledger model, and its formalization in terms of small-step semantic [30]. A UTxO ledger, which stands for *Unspent Transaction Outputs*, records entries (i.e. *transaction outputs*) in a *UTxO set*. This model was first introduced in the BitCoin system [23], and currently in use by the Cardano [19] and Ergo [12]. A UTxO entry contains some funds, together with a specification of what kinds of transactions are allowed to "spend" this record, e.g. only ones signed by a particular key. Applying a transaction to a UTxO set only adds of removes entries, never modifying them. This makes performing formal analysis on this model more tractable in many cases. In particular, the outcome of a transaction application is not affected by the order of transaction application, as we demonstrate in this work.

Cynthia Kop and Helida Salles Santos (Eds); 19th Workshop on
Logical and Semantic Frameworks with Applications (LSFA'24)
EPTCS 421, 2025, pp. 1–20, doi:10.4204/EPTCS.421.1

An *extended* UTxO (EUTxO) ledger is a UTxO-style ledger that supports the use of *smart contracts*. EUTxO smart contracts are pieces of user-defined code that specify permissions for certain transaction actions. In this work, we omit most of the details of this mechanism as it is not necessary for our construction. However, we do make use of a stateful contract computation model called a *structured contract framework* (SCF) [30] that relies on the possibility of composing sophisticated permissions. This framework specifies the condition that any valid transaction applied to the ledger state will result in an update for the program state encoded in that ledger state that is in accordance with the program specification. This condition is used as the definition of a a program being implemented on the ledger, including both consolidated and distributed programs. This approach only formalizes a single "correct" step of a program, both in its ledger implementation and specification.

Trace-based properties [1] are predicates on infinite sequences of program states used to reason about the correctness of stateful programs. Unlike the structured contract formalism, this approach allows for reasoning about multiple program steps at a time, as it studies program executions of infinite length. As is, in order to demonstrate the correctness of the (single step) program state update, the SC approach requires making certain assumptions, which are related to the behavior of the UTxO ledger program itself. These assumptions can only be expressed in terms of subsets of ledger traces generated in a specific way, i.e. a specific trace-based property. Moreover, the basic definitions of the SCF formalism have not been used to to reason about traces or express any sophisticated guarantees of correct behavior.

We make the observation that correctly-implemented programs, then, can be shown to behave correctly only when their behavior is considered on ledger traces that are themselves generated from a valid start state, and according to the ledger program specification. In this work, we take a principled approach to formalizing this idea, while filling these gaps in analysis of the ledger program and the implementations of other programs on it. In particular, we use ideas from trace-based behavior analysis, graph theory, category theory, and topology, to do the following analysis:

(i) we define what it means for a ledger or contract trace to have the property of being *valid*, defined in terms of small-steps specifications and initial states;

(ii) we define a category of simple graphs with fixed initial vertices, paths in which represent valid execution traces, and the maps are partial sieve-defined homomorphisms between them;

(iii) we demonstrate that infinite paths in the graphs of this category form an ultrametric space [20]. As a corollary, we obtain that any structured contract induces a non-expanding (and therefore continuous) map from ledger traces to structured contract traces;

(v) we prove a number of classic UTxO ledger safety properties within our framework , including transaction commutativity for valid permutations, and replay protection.

The value in defining the category in (ii) is that it formalizes the relation of being "correctly implemented on" between *a stateful program* and the *ledger program which executes it* in a way that aligns with existing trace-based behavior analysis techniques. Formalizing the relation between properties of these programs allows us to derive useful results about ledger behavior based on the properties of implemented programs, and vice versa. In particular, a direct consequence of the nature of our construction is that a program property of the form "a specific bad thing never happens" (i.e., a safety property) is satisfied by all valid execution traces generated from a correct implementation of that program on the ledger. Finally, we note that the results in (v) are proved, instead of being treated as an assumption as is done in some existing work on implementing programs on a UTxO ledger in order to prove the correctness of its implementation [29]. Since these trace properties cannot be expressed as predicates on an a specific state, this required the additional machinery we introduce here to formalize.

2 Small-step specifications and the ledger model

We present an overview of the types and semantics employed in our specifications, on which we later base our analysis.

2.1 Small-step specifications

Small-step semantics are specified in terms of of atomic steps from which all other (composite) steps can be built. We establish the following notation for our upcoming specifications:

Definition 2.1.1. A *small-step transition system* TRANS is given by a subset $\mathsf{TRANS} \subseteq \mathsf{Env} \times \mathsf{State} \times \mathsf{Input} \times \mathsf{State}$, also denoted $e \vdash s \xrightarrow[\text{TRANS}]{i} s'$

Each component in a quadruple $(e,\ s,\ i,\ s')$ has the following role: (i) $e \in \mathsf{Env}$ is an *environment*; (ii) $s \in \mathsf{State}$ is a state to which an *input* $i \in \mathsf{Input}$ is applied, or a *starting state*; (iii) $s' \in \mathsf{State}$ is a state representing, for the given environment, a result of the input application to the starting state, or the *end state*. It is possible that a given state has no valid transitions out of it, or that a given input is not included in valid transition 4-tuple.

Given a transition $(e,\ s,\ i,\ s') \in \mathsf{TRANS}$, the pair $(e,\ i)$ of an environment and an input make up a *transition label* of TRANS. The difference between the input and the environment is in the source of the data. This convention comes from the specification of the Cardano ledger transition system [10]. The user issues the input, e.g., a transaction. The environment is sourced from the transaction-containing blocks (which we do not model here). For example, the consensus and block production mechanisms keep track of the current time, which is specified by the environment in our model.

2.2 Ledger transition system

A ledger is a stateful program that implements one operation: the application of a transaction to the current ledger state. In the Extended UTxO (EUTxO) ledger model, the state is made up of a set of *unspent transaction outputs*, called the UTxO set. A single UTxO entry (unspent output) consists of a unique identifier together with an output. The output contain data, assets, and code (i.e., a smart contract, which specifies what transactions have are permitted to remove this output from the UTxO set). UTxO entries are immutable, can can only be either added or removed from the UTxO set. We give an overview of the EUTxO state and transaction structure below, leaving abstract any types and definitions that are not relevant for the examples we present later. A complete specification is available in [30]. Nonstandard notation we use here is specified in Figure 1.

$\star : \{\star\}$	denotes the one-element set, and its one inhabitant
$Q : \mathsf{Set}\ (A)$	Q is a set of elements of type A
$Key \mapsto Value \subsetneq [(Key,\ Value)]$	finite map

Figure 1: Non-standard map operators

UTxO set. A *UTxO set* constitutes the state of a ledger model. It is given by a finite map (finite associative array) $(\mathsf{ByteString}, \mathbb{N}) \mapsto \mathsf{Output}$ The key in the UTxO finite map is called an *output reference*.

Slot number. To represent blockchain time, a natural number is used, which we call a *slot number* (or just *slot*), with $\mathsf{Slot} = \mathbb{N}$. A pair of slot numbers represents an interval which includes the first (starting) slot of the pair, but excludes the second (end) slot.

Transaction. Updates that a user wants to make to the UTxO set are specified in a Tx and Input data structures

$$\begin{aligned}\mathsf{Tx} = (&\mathsf{inputs} : \mathsf{Set}\ (\mathsf{Input}),\\ &\mathsf{outs} : \mathsf{List}[\mathsf{Output}],\\ &\mathsf{validityInterval} : (\mathsf{Slot}, \mathsf{Slot}),\\ &\mathsf{additionalData} : \mathsf{AdditionalData})\end{aligned} \qquad \begin{aligned}\mathsf{Input} = (&\mathsf{outputRef} : (\mathsf{ByteString}, \mathbb{N}),\\ &\mathsf{output} : \mathsf{Output},\end{aligned}$$

A transaction tx consists of (i) a set $\mathsf{inputs}(tx)$ of inputs (pointers to entries in the UTxO set), (ii) outputs $\mathsf{outs}(tx)$ (which will be added as values to the UTxO set, accompanied by appropriate unique identifier pointers), (iii) a pair of slots $\mathsf{validityInterval}(tx)$ representing the interval of time during which this transaction can be processed. We include the field additionalData in order to suggest how to extend this model to represent the full EUTxO ledger model [30]. Also, Tx is equipped with a hash function $h : \mathsf{Tx} \to \mathsf{ByteString}$. that is used, for a given transaction tx, to generate the unique identifier $(\mathsf{hash}\ tx, ix)$. This identifier, called an output reference, is used when adding output $o_{ix} \in \mathsf{outs}\ tx$ with index ix (in the list of outputs of tx) to the UTxO set.

The following gives the type of the ledger transition system $\mathsf{LEDGER} \subseteq \mathsf{Slot} \times \mathsf{UTxO} \times \mathsf{Tx} \times \mathsf{UTxO}$ We require any member (q,u,t,u') of LEDGER to satisfy the following constraints: (i) a transaction t has at least one input; (ii) a slot q is within the validity interval of transaction t; (iii) all transaction inputs exist in the UTxO set u. We define a function which checks these constraints, $\mathsf{checkTx} : \mathsf{Slot} \times \mathsf{UTxO} \times \mathsf{Tx} \to \mathsf{Bool}$ by the following rule

$$\mathsf{checkTx}\ (q,u,t) := (\mathsf{inputs}\ t \neq \varnothing) \wedge (q \in \mathsf{validityInterval}(t))$$

$$\wedge(\forall i \in \mathsf{inputs}\ t,\ (\mathsf{outputRef}\ i) \mapsto (\mathsf{output}\ i) \in u) \wedge (\mathsf{additionalChecks}\ (q,u,t))$$

Again, the field additionalChecks is included in order to ensure the model can be extended to full EUTxO. The function checkTx specifies, for a given triple $(q,\ u,\ t)$, the conditions that must be satisfied in order for there to exist a u' with $(q,u,t,u') \in \mathsf{LEDGER}$. When it exists, $u' \in \mathsf{UTxO}$ is computed as follows: (i) UTxO entries in u that correspond to the inputs of the transaction t must be removed; (ii) entries constructed from the outputs of the transaction t must be added to u. We call the functions getORefs and mkOuts respectively (defined in Figure 2). The following rule ApplyTx defines the membership of (q,u,t,u') in the LEDGER relation:

$$u' := (u \setminus \mathsf{getORefs}(t)) \cup \mathsf{mkOuts}(t)$$

$$\text{ApplyTx}\ \frac{\mathsf{checkTx}\ (q,u,t)}{q \vdash u \xrightarrow[\text{LEDGER}]{t} u'} \tag{1}$$

2.3 Structured contracts

To motivate the category we define later, we specify of what it means for a *stateful contract* to be implemented on the UTxO ledger. User-defined code in the EUTxO model is not stateful, and can only take the

$$
\begin{aligned}
\mathsf{toMap} &: \mathsf{Ix} \to [\mathsf{Output}] \to (\mathsf{Ix} \mapsto \mathsf{Output}) \\
\mathsf{toMap}\ ix\ \{\} &= [\,] \\
\mathsf{toMap}\ ix\ [u;\ outs] &= \{\ ix \mapsto u\ \} \cup \{\ (\mathsf{toMap}\ (ix+1)\ outs)\ \}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{mkOuts} &: \mathsf{Tx} \to \mathsf{UTxO} \\
\mathsf{mkOuts}\ tx &= \{\ (tx,\ ix) \mapsto o\ \mid\ (ix \mapsto o) \in\ \mathsf{toMap}\ 0\ (\mathsf{outs}\ tx)\ \}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{getORefs} &: \mathsf{Tx} \to \mathsf{Set}\ (\mathsf{OutputRef}) \\
\mathsf{getORefs}\ tx &= \{\ \mathsf{outputRef}\ i\ \mid\ i\ \in\ \mathsf{inputs}\ tx\ \}
\end{aligned}
$$

Figure 2: Auxiliary UTxO functions

form of boolean predicates on transaction data. To reason about stateful contracts, we use the structured contract model [30]. Let STRUC be a program expressed in terms of the small-steps semantics. The state of this contract is represented (encoded) on the ledger in some specific way. This encoding is specified in terms of a (partial) projection function that computes the contract state for a given ledger state (or fails). Similarly, for a given transaction, the input to STRUC can be computed. The structured contract specification STRUC is said to be *implemented correctly* whenever, for a given ledger state and transaction, the structured contract state (encoded within that ledger state) is updated by the input (encoded withing the transaction) in accordance with the STRUC specification.

The specification STRUC, together with the projection functions and a proof the the correctness of the implementation, constitutes a structured contract. Note here that we do not show the exact details of running user-defined code predicates on transaction data. In a full model, we would fill in these details by specifying the required additionalData and additionalChecks. Depending on the choice of projection functions, this model can be used to represent both distributed and consolidated contracts [30]. We formalize,

Definition 2.3.1. Suppose STRUC is small-step transition system $\mathsf{STRUC} \subseteq \{*\} \times \mathsf{State} \times \mathsf{Input} \times \mathsf{State}$ and let we have a partial function $\pi : \mathsf{UTxO} \rightharpoonup \mathsf{State}$ and a (total) function [1] $\kappa : \mathsf{Tx} \to \mathsf{Input}$ such that

$$
\frac{(u \in \mathrm{Def}\,\pi)\ \wedge\ \left(\ q\ \vdash\ u\ \xrightarrow[\text{LEDGER}]{t}\ u'\ \right)}{(u' \in \mathrm{Def}\,\pi)\ \wedge\ \left(\ *\ \vdash\ \pi u\ \xrightarrow[\text{STRUC}]{\kappa t}\ \pi u'\ \right)} \tag{2}
$$

The triple $(\mathsf{STRUC}, \pi, \kappa)$ is called a *structured contract* for LEDGER.

Following existing EUTxO design, no block-level data is exposed to a smart contract, such as a current slot number, so a singleton $\{*\}$ is the type of a structured contract specification environment. An example of a very basic program that can be defined as a structured contract would be an NFT, or a non-fungible (i.e. unique) token [30]. The state of an NFT contract is the total quantity of this type of token in the UTxO set, and the constraint in the update rule of that state is that the updated (end state) quantity may not exceed 1.

[1] A total function f from X to Y means everywhere-defined function on X, while a partial function f from X to Y means a total function from a subset $\mathrm{Def}(f) \subseteq X$ to Y.

As is, structured contracts give only the guarantee that each program step will be correct, both in the specification and in the ledger implementation. In the rest of this work, define the space of valid program executions constructed from its small-step specification in a way that aligns with existing definitions and terminology in the field. This allows for expressing (and proving adherence to) a wider range of properties, and formalizing the relationship between valid executions of ledger programs and executions of their implementations.

3 Traces

In [1], the collection of *traces* (or *executions*) of a program with states of type S is a collection of infinite lists of states. Taking $S = \mathsf{UTxO}$, we get a set of all UTxO traces, and taking $S = \mathsf{State}$, we get the set of all contract state traces. In both cases we are dealing with *arbitrary lists of states*. However, we are interested in reasoning about traces that are generated according to their specification. Therefore, we need to define appropriate subsets of infinite lists, which we refer to as *valid traces*:

Definition 3.0.1 (Valid ledger and structured contract traces)**.**

(i) Fix subsets $\mathsf{UTxO}_0 \subseteq \mathsf{UTxO}, \mathsf{Slot}_0 \subseteq \mathsf{Slot}$ called *valid initial UTxO states* and *valid initial slots*, respectively. A *set of valid* LEDGER *traces* is

$$\begin{aligned}\mathsf{Trc}(\mathsf{LEDGER}) := \{(u_0,u_1,\ldots) \in \mathsf{List}_\infty[\mathsf{UTxO}] \mid \forall j \geq 0,\ \exists\, (q_j,u_j,t_j,u_{j+1}) \in \mathsf{LEDGER}: \\ (q_0,u_0) \in \mathsf{Slot}_0 \times \mathsf{UTxO}_0,\ q_0 \leq q_1 \leq \ldots\}\end{aligned}$$

Note that it is not enough to require that each 4-tuple is in LEDGER, the slots must appear in the increasing order.

(ii) Let $(\mathsf{STRUC},\ \pi,\ \kappa)$ be a structured contract for LEDGER. We say that $\mathsf{State}_0 \subseteq \mathsf{State}$ is a set of *valid initial* STRUC *states* if all valid initial UTxO states are mapped to State_0 via the partially defined map π, i.e. $\mathsf{UTxO}_0 \subseteq \mathrm{Def}\,\pi$ and $\pi(\mathsf{UTxO}_0) \subseteq \mathsf{State}_0$. A *set of valid* STRUC *traces* is

$$\mathsf{Trc}(\mathsf{STRUC}) := \{\ (s_0,s_1,\ldots) \in \mathsf{List}_\infty[\mathsf{State}] \mid \forall j \geq 0,\ (*,\ s_j,\ i_j,\ s_{j+1}) \in \mathsf{STRUC},\ s_0 \in \mathsf{State}_0\ \}$$

Since the environment type for structured contracts is $\{*\}$, there is no restriction on it. For both Trc(LEDGER) and Trc(STRUC) we refer to lists of quadruples corresponding to valid traces in LEDGER and STRUC, respectively, as *lifts* of that traces.

4 Graphs and sieve-defined homomorphisms

The specification LEDGER can be used to generate sequences of the form (q,u,t_0,u_0), (q,u_0,t_1,u_1), ..., such that for each triple (q,u_i,t_i), $\mathsf{checkTx}\ (q,u_i,t) = \mathsf{True}$. This process gives rise to a *simple graph*[2] Λ, whose vertices are triples (q,u,t) satisfying checkTx, and edges connecting any two (q,u,t) and (q',u',t') whenever $(q,u,t,u') \in \mathsf{LEDGER}$. Similarly, a structured contract specification STRUC defines another simple graph Γ on triples $(*,s,i)$. We formalize the relation between simple graphs in general, and in particular, between graphs related via implementation, such as LEDGER and STRUC.

Note that in this section and onwards, we use standard terms and definitions of category theory, as described in existing works [21, 4, 5].

[2]By a simple graph we mean a directed graph in which for any two vertices x and y there is at most one edge $e : x \to y$. Single loops at vertices are also allowed.

4.1 Simple graphs and sieve-defined homomorphisms

In general, a homomorphism of directed graphs is a pair of maps: one for vertices and one for edges, such that the map of edges respects their domains and codomains. In the case of a simple graphs, a map for vertices completely defines the homomorphism.

Let G be the graph generated from LEDGER, and G' be the graph generated from G'. The partially defined function $\pi : \mathsf{UT}\times\mathsf{O} \rightharpoonup \mathsf{State}$ and total function $\kappa : \mathsf{T}\times \to \mathsf{Input}$ define a partial map φ from the set of vertices of G to the set of vertices of G', such that the domain of definition of φ, denoted $\operatorname{Def}\varphi$, is *upward closed*. That is, for an edge $(q,u,t) \to (q',u',t')$ in G, and $(q,u,t) \in \operatorname{Def}\varphi$, then $(q',u',t') \in \operatorname{Def}\varphi$. This follows immediately from the definition of structured contract. Moreover, φ defines a homomorphism of simple graphs from $\operatorname{Def}\varphi$ to G': $((q,u,t) \to (q',u',t')) \mapsto ((*,\pi u,\kappa t) \to (*,\pi u',\kappa t'))$.

This can be made precise as the following definition,

Definition 4.1.1. Let $G = (V,E)$ be a graph. A subset $S \subseteq V$ is called a *sieve*, whenever any edge starting in S necessary ends in S. That is, if $e : v_1 \to v_2 \in E$ and $v_1 \in S$, then $v_2 \in S$. Equivalently, every path in G that starts in S also ends in S.

The following are trivial examples of sieves:

Example 4.1.2. For any graph $G = (V,E)$ the set of all vertices V and any sink in G are sieves. Also, any intersection of sieves is a sieve.

We can now define the class of graph homomorphisms that define the implementation relation between two specifications in graph-theoretic terms:

Definition 4.1.3. Let $G = (V,E)$ and $G' = (V',E')$ be simple graphs. A partial map $\varphi : V \rightharpoonup V'$, whose domain of definition $\operatorname{Def}\varphi$ is a sieve in G, is called a *partially sieve-defined homomorphism*, if φ extends to a homomorphism of graphs from the full subgraph of G on $\operatorname{Def}\varphi$ to the graph G'. Such map will be denoted $\varphi : G \rightharpoonup G'$ and $\operatorname{Def}\varphi$ will also denote the corresponding full subgraph in G.

All everywhere-defined homomorphisms of simple graphs are also partial sieve-defined homomorphism of simple graphs. In particular, the identity homomorphism is a trivial example of this. The following lemma is needed to define the composition of sieve-defined homomorphisms:

Lemma 4.1.4 (Sieve of a sieve is a sieve)**.** Let $G = (V,E)$ be a graph and S a sieve in G. Suppose S' is a sieve in a full subgraph $\overline{S}$ of G defined by S. Then S' is a sieve in G.

Proof. Let $e : v \to v'$ be an edge in G such that $v \in S'$. Since $S' \subseteq S$, then $v \in S$. Hence, v' belongs to S, as S is a sieve in G. Moreover, the edge e is in the full subgraph $\overline{S}$ of G on S. By the assumption, S' is a sieve in $\overline{S}$, so v' belongs to S'. □

Recall the composition of partially defined functions between sets. Suppose $f : X \rightharpoonup Y$ and $g : Y \rightharpoonup Z$ are partially defined functions. We define their composition to be the partially defined function $g \circ f : X \rightharpoonup Z$, $\operatorname{Def}(g \circ f) = \operatorname{Def} f \cap f^{-1}(\operatorname{Def} g)$ and $(g \circ f)x = g(f(x))$ if $x \in \operatorname{Def}(g \circ f)$.

Since any partial sieve-defined homomorphism of simple graphs is completely defined by a map on its vertices, we define the composition of partial sieve-defined homomorphisms in a similar way to sets.

Lemma 4.1.5. Partial sieve-defined homomorphisms of simple graphs are closed under composition.

Proof. Given $f : G \rightharpoonup G'$ and $g : G' \rightharpoonup G''$ it is enough to prove that $\operatorname{Def}(g \circ f) = \operatorname{Def} f \cap f^{-1}(\operatorname{Def} g)$ is a sieve in $\operatorname{Def} f$. Suppose $e : v \to v'$ is an edge in $\operatorname{Def} f$ such that $v \in \operatorname{Def}(g \circ f)$. The edge $f(e) : f(v) \to f(v')$ then belongs to G', where $f(v) \in \operatorname{Def} g$. Since $\operatorname{Def} g$ is a sieve, $f(v')$ also belongs to $\operatorname{Def} g$. Therefore, $v' \in \operatorname{Def} f \cap f^{-1}(\operatorname{Def} g)$. We then get that $\operatorname{Def}(g \circ f)$ is a sieve in $\operatorname{Def} f$. Since a sieve of a sieve is a sieve, $g \circ f : G \rightharpoonup G''$ is a partial sieve-defined homomorphism of simple graphs. □

The above information motivates us to introduce a category $\mathbf{Graph}^\sharp$ of all simple graphs and partial sieve-defined homomorphisms.

4.2 Graphs with distinguished initial vertices

The definition of the set of valid traces for LEDGER and STRUC motivates us to add a subset of *initial vertices* $\mathring{V} \subseteq V$ to each simple graph $G = (V,E)$. If $f : G \rightharpoonup G'$ is a partial sieve-defined homomorphism of simple graphs, then we require $\mathring{V} \subseteq \mathrm{Def}\, f,\ f(\mathring{V}) \subseteq \mathring{V}'$, i.e., each partial sieve-defined homomorphism should be defined over initial vertices and preserve them. We generalize this situation and define an appropriate category.

Definition 4.2.1. Let $\mathbf{Graph}^\sharp_*$ be a category whose objects are pairs $(G;\mathring{V})$, where $G = (V,E)$ is a simple graph. Let $\mathring{V} \subseteq V$ be a set of *initial vertices*, so that the morphisms of $\mathbf{Graph}^\sharp_*$ from $(G;\mathring{V})$ to $(G';\mathring{V}')$, denoted $((G;\mathring{V}),(G',\mathring{V}'))^\partial$, are (partial) maps $f \in \mathbf{Graph}^\sharp(G,G')$ such that $\mathring{V} \subseteq \mathrm{Def}\, f, f(\mathring{V}) \subseteq \mathring{V}'$. The composition and identities are inherited from ones in $\mathbf{Graph}^\sharp$. The set of everywhere-defined maps $f : G \to G'$ such that $f(\mathring{V}) \subseteq \mathring{V}'$ we denote by $((G;\mathring{V}),(G';\mathring{V}'))$.

If a set of initial vertices is clear from the context, we can abbreviate $(G;\mathring{V})$ to just G, still considering it as an object of $\mathbf{Graph}^\sharp_*$. Obviously, there is an inclusion $((G;\mathring{V}),(G',\mathring{V}')) \subseteq ((G;\mathring{V}),(G',\mathring{V}'))^\partial$.

Let us consider the following covariant representable functor, which we use in an upcoming example (where the above inclusion becomes an equality),

$$(\mathbb{N},\_) : \begin{cases} \mathbf{Graph}^\sharp_* \longrightarrow \mathbf{Set} \\ G \longmapsto (\mathbb{N},G) \\ (f : G \rightharpoonup G') \longmapsto (f \circ \_ : (\mathbb{N},G) \to (\mathbb{N},G')) \end{cases}$$

We will use the following example to represent and study paths in $\mathbf{Graph}^\sharp_*$ as executions (or traces) programs:

Example 4.2.2. The natural ordering on the set of natural numbers $\mathbb{N}$ induces a simple graph $0 \to 1 \to \ldots$, which will be also denoted by $\mathbb{N}$. By default, the set of initial vertices for $\mathbb{N}$ is set to be $\{0\}$, i.e., $(\mathbb{N};\{0\}) \in \mathbf{Graph}^\sharp_*$. Moreover, given any $f : (\mathbb{N};\{0\}) \rightharpoonup (G;\mathring{V})$, we get that $0 \in \mathrm{Def}\, f$ and $\mathrm{Def}\, f$ is a sieve in G. Since there exists a unique path in $\mathbb{N}$ from 0 to any $n > 0$, any $n \in \mathbb{N}$ also belongs to N. Therefore, $\mathrm{Def}\, f = \mathbb{N}$, i.e., any partial sieve-defined homomorphism from $(\mathbb{N};\{0\})$ to $(G;\mathring{V})$ is everywhere defined: $((\mathbb{N};\{0\}),(G;\mathring{V})) = ((\mathbb{N};\{0\}),(G;\mathring{V}))^\partial$ or, skipping the sets of initial vertices, $(\mathbb{N},G) = (\mathbb{N},G)^\partial$. Any $f : \mathbb{N} \to G$ models an infinite path $v_0 \to v_1 \to \ldots$ in the simple graph G, where $v_i = f(i)$ and $v_0 \in \mathring{V}$. Of course, $(\mathbb{N},G) = \varnothing$, if G is acyclic and finite.

Remark 4.2.3. When $G = (V,E)$ is a complete graph (including single loops at vertices), we identity $(\mathbb{N},G)$ with a set $\langle\!\langle V \rangle\!\rangle := \{\vec{v} = (v_0,v_1,\ldots) \mid v_i \in V\}$ of infinite lists of elements of V. These arbitrary infinite lists of elements of V are exactly the execution traces of program with states in V [1].

4.3 Graphs of LEDGER, STRUC, and their traces

Recall that applying the functor $(\mathbb{N},\_)$ to a simple graph G returns the set of infinite paths in G. We use Example 4.2.2 to define the abstract notion of a *trace of a map*, which corresponds to the set of valid traces (considered as infinite paths in a state transition graph) that can be generated using this functor.

Definition 4.3.1. Let $f : (G;\mathring{V}) \rightharpoonup (G';\mathring{V}')$ be a morphism in $\mathbf{Graph}^\sharp_*$. An image $\mathrm{Im}\, f_*$ of a postcomposition function $f_* = f \circ \_ : (\mathbb{N},G) \to (\mathbb{N},G')$ is called a *trace* of f. We'll denote the trace of f by $\mathsf{Trc}(f)$.

Unfolding the above definition, we get that the trace of $f : (G;\mathring{V}) \rightharpoonup (G';\mathring{V}')$ consists of infinite paths $v'_0 \to v'_1 \to v'_2 \to \ldots$ in G', where $v'_0 \in \mathring{V}'$, which has lifts in G, i.e., an infinite path $v_0 \to v_1 \to v_2 \to \ldots$ in G, where $v_0 \in \mathring{V}$, with $f(v_0) = v'_0$, $f(v_1) = v'_1$, $f(v_2) = v'_2$, ... etc.

The definition of a valid LEDGER trace can be restated in terms of infinite paths in graphs. Let $\Lambda = (V,E)$ be a graph whose set of vertices V consists of triples (q,u,t) satisfying the condition checkTx, and that there is an edge $(q,u,t) \to (q',u',t')$ if and only if $(q,u,t,u') \in \mathsf{LEDGER}$ and $q \leq q'$. The constructed graph Λ is simple, as u' is uniquely defined. Moreover, we specify the set of initial vertices to be $\mathring{V} := \{(q,u,t) \in \mathsf{Slot}_0 \times \mathsf{UTxO}_0 \times \mathsf{Tx} \mid \mathsf{checkTx}(q,u,t)\}$

Similarly, we define a simple graph $\Lambda' = (V',E')$, where V' is a set of all $u \in \mathsf{UTxO}$ such that there are $q \in \mathsf{Slot}$ and $t \in \mathsf{Tx}$ with $\mathsf{checkTx}(q,u,t) = \mathsf{True}$ and there is an edge $u \to u'$ if and only if there are $q \in \mathsf{Slot}$ and $t \in \mathsf{Tx}$ with $(q,u,t,u') \in \mathsf{LEDGER}$. Simply speaking, Λ' is a "projection" of Λ on its UTxO-component. For the set of initial vertices in Λ' we take $\mathring{V}' := V' \cap \mathsf{UTxO}_0$. The projection map $\varphi : V \to V'$ given by $\varphi(q,u,t) = u$ induces a morphism $\varphi : (\Lambda;\mathring{V}) \rightharpoonup (\Lambda';\mathring{V}')$ in $\mathbf{Graph}^\sharp_*$. Now, we apply the functor $(\mathbb{N}, \_)$ to the morphism $\varphi : \Lambda \rightharpoonup \Lambda'$ we obtain a postcomposition function $\varphi_* = \varphi \circ \_ : (\mathbb{N},\Lambda) \to (\mathbb{N},\Lambda')$. Simply speaking, φ_* takes an infinite path $(q_0,u_0,t_0) \to (q_1,u_1,t_1) \to (q_2,u_2,t_2) \to \ldots$ in Λ, where $(q_0,u_0,t_0) \in \mathring{V}$, to an infinite path $u_0 \to u_1 \to u_2 \to \ldots$ in Λ'.

Taking $\varphi = \mathsf{LEDGER}$, and recalling the definition of the set of valid LEDGER traces we obtain that $\mathsf{Trc}(\mathsf{LEDGER}) = \operatorname{Im} \varphi_*$

Let $(\mathsf{STRUC}, \pi, \kappa)$ be a structured contract implemented on LEDGER. Then, a similar construction exists for STRUC. Let $\Gamma = (W,F)$ be a graph whose vertices are pairs $(s,i) \in \mathsf{State} \times \mathsf{Input}$ such that there exists $s' \in \mathsf{State}$ with $(*,s,i,s') \in \mathsf{STRUC}$ and there exists an edge $(s,i) \to (s',i')$ if and only if $(*,s,i,s') \in \mathsf{STRUC}$. In the simple graph Γ we specify the set of initial vertices

$$\mathring{W} := \left\{(s,i) \in \mathsf{State}_0 \times \mathsf{Input} \mid \exists\, s' \in \mathsf{State} :\ (*,s,i,s') \in \mathsf{STRUC}\right\}.$$

STRUC induces another graph $\Gamma' = (W',F')$ in the same way that Λ is induced for LEDGER. Here, W' is a set of all $s \in \mathsf{State}$ such that there are $i \in \mathsf{Input}$ and $s' \in \mathsf{State}$ with $(*,s,i,s') \in \mathsf{STRUC}$ and there is an edge $s \to s'$ if and only if there are $i,i' \in \mathsf{Input}$ with $(s,i),(s',i') \in W'$ and $(*,s,i,s') \in \mathsf{STRUC}$. The set of initial vertices of Γ is $\mathring{W}' := W' \cap \mathsf{State}_0$. Similarly to $\varphi : (\Lambda;\mathring{V}) \rightharpoonup (\Lambda';\mathring{V}')$, the projection map $\psi : W \to W'$ induces a morphism $\psi : (\Gamma;\mathring{W}) \rightharpoonup (\Gamma';\mathring{W}')$ in $\mathbf{Graph}^\sharp_*$. Again, applying the functor $(\mathbb{N}, \_) : \mathbf{Graph}^\sharp_* \to \mathbf{Set}$ to $\psi : \Gamma \rightharpoonup \Gamma'$, we obtain a postcomposition function $\psi_* = \psi \circ \_ : (\mathbb{N},\Gamma) \to (\mathbb{N},\Gamma')$, which takes an infinite path $(s_0,i_0) \to (s_1,i_1) \to (s_2,i_2) \to \ldots$ in Γ, where $(s_0,i_0) \in \mathring{W}$, to an infinite path $s_0 \to s_1 \to s_2 \to \ldots$ in Γ'. Taking $\psi = \mathsf{STRUC}$, and recalling the definition of the set of valid STRUC traces we obtain $\mathsf{Trc}(\mathsf{STRUC}) = \operatorname{Im} \psi_*$.

Maps $\pi : \mathsf{UTxO} \rightharpoonup \mathsf{State}$ and $\kappa : \mathsf{Tx} \to \mathsf{Input}$ induce well-defined morphisms $\sigma(q,u,t) = (\pi u, \kappa t)$ and $\sigma'(u) = \pi u$ in $\mathbf{Graph}^\sharp_*$ such that the following diagram commutes:

$$\begin{array}{ccc} (\Lambda;\mathring{V}) & \xrightarrow{\ \sigma\ } & (\Gamma;\mathring{W}) \\ {\scriptstyle\varphi}\big\downarrow & & \big\downarrow{\scriptstyle\psi} \\ (\Lambda';\mathring{V}') & \xrightarrow{\ \sigma'\ } & (\Gamma';\mathring{W}') \end{array}$$

This shows that when π and κ specify a correct implementation of the STRUC program on the ledger, a (valid) state trace $\vec{v} \in \mathsf{Trc}(\mathsf{LEDGER})$, corresponding to a specific lift (i.e., a path in graph that contains both state and input information), maps to a (valid) trace of $\vec{w} \in \mathsf{Trc}(\mathsf{STRUC})$. Moreover, $\vec{w}$ necessarily corresponds to the lift that is obtained by applying the function induced by π and κ to the lift of $\vec{v}$. In other words, $\vec{v}$ and $\vec{w}$ are "generated" by corresponding (via κ) sequences of inputs. This conclusion is the key consequence of fulfilling the proof obligation 2.3.1 required as part of instantiating $(\mathsf{STRUC}, \pi, \kappa)$.

Our goal is now to prove that the above square induces a well-defined continuous postcomposition function $\overline{\pi} : \mathsf{Trc}(\mathsf{LEDGER}) \to \mathsf{Trc}(\mathsf{STRUC})$, thereby ensuring that correct program implementations are also well-behaved with respect to certain kinds of properties. To achieve this goal we will study a specific kind of metric.

4.4 Ultrametric spaces and traces

Sets of morphisms $(\mathbb{N}, G)$ in $\mathbf{Graph}_*^\sharp$ also carry an additional structure: they are metric spaces. A metric d on the set $(\mathbb{N}, G)$ is $d(\vec{a}, \vec{b}) := \inf\left\{2^{-k} \mid k \geq 0, a_i \neq b_i\right\}$. The function d also satisfies a strengthened version of the triangle inequality: $d(\vec{a}, \vec{b}) \leq \max\{d(\vec{a}, \vec{c}), d(\vec{c}, \vec{b})\}$.

This metric has been previously defined for arbitrary execution traces of arbitrary programs[1]. In this work, we apply this definition to the space of only valid execution traces. Spaces admitting the type of metric given above are said to be *ultrametric* [20]. Ultrametric spaces have the following properties, which we will later make use of:

Proposition 4.4.1. Let (X, d) be an ultrametric space. Then, (i) any triangle in X is isosceles, i.e., for any $x, y, z \in X$, two of the numbers $d(x,y), d(y,z), d(x,z)$ are equal and greater than the third one; (ii) every point inside an open ball is its center; (iii) if two open balls intersect, then one is contained in another; (iv) every open ball of a positive radius is a closed set.

From the above proposition following properties of the ultrametric space can be derived $((\mathbb{N}, G), d)$: (i) d takes values in a countable set $\{0, 1, \frac{1}{2}, \frac{1}{4}, \ldots\}$; (ii) the whole space coincides with a closed unit ball, i.e., $(\mathbb{N}, G) = \overline{B}(\vec{v}, 1)$ for any $\vec{v} \in (\mathbb{N}, G)$; (iii) for any $\vec{u} \in (\mathbb{N}, G)$ and $r > 0$, $B(\vec{u}, r) = B(\vec{u}, 2^{-\underline{r}}) = B(\vec{u}, 2^{-(\underline{r}+1)})$, where $\underline{r} = \lfloor -\log_2 r \rfloor$, i.e., there are only a countable number of balls with a center $\vec{u}$; (iv) for any $\vec{u} \in (\mathbb{N}, G)$ and $r > 0$, $B(\vec{u}, r) := \{\vec{v} \in (\mathbb{N}, G) \mid \vec{v}[n] = \vec{u}[n]\}$, where $\vec{u}[n]$ is a head of the path of $\vec{u}$ of length n in G, i.e., a ball of radius r with a center $\vec{u}$ consists of infinite paths in G, that share an $\underline{r}$- head with $\vec{u}$. From now we will treat $(\mathbb{N}, G)$ as an ultrametric space with the described above ultrametric d.

The following proposition states that morphisms in $\mathbf{Graph}_*^\sharp$ induce functions between sets of infinite paths that are, in fact, continuous maps of ultrametric spaces. To prove this, we will use the fact that the induced map f_* simply applies f to each individual state in the infinite path.

Proposition 4.4.2. If $f : G \rightharpoonup G'$ is a morphism in $\mathbf{Graph}_*^\sharp$, then $f_* : (\mathbb{N}, G) \to (\mathbb{N}, G')$ is a continuous map.

Proof. For arbitrary $\vec{u}$ in $(\mathbb{N}, G)$, and $n \geq 0$, take any $\vec{v} \in B(f_*\vec{u}, 2^{-n})$. Then $\vec{v}$ has form $fu_0 \to \ldots \to fu_n \to v_{n+1} \to \ldots$. For any $\vec{w} \in B(\vec{u}, 2^{-n})$, its image under f_* is $fu_0 \to \ldots \to fu_n \to fw_{n+1} \to \ldots$. Hence, $f\vec{w} \in B(f_*\vec{u}, 2^{-n})$. and f_* is continuous. □

In order to study ultrametric structure in $\mathbf{Graph}_*^\sharp$, we define the following category:

Definition 4.4.3. A *non-expanding map* between ultrametric spaces (X, d_X) and (Y, d_Y) is a continuous map $f : X \to Y$ such that $d_Y(f(x_1), f(x_2)) \leq d_X(x_1, x_2)$ for any $x_1, x_2 \in X$. A category of ultrametric spaces and non-expanding maps we denote **UMet**.

We now use the **UMet** category to express that all maps between sets of infinite paths in objects of our category $\mathbf{Graph}_*^\sharp$ (i.e., simple graphs) are continuous and non-expanding:

Proposition 4.4.4. A functor $(\mathbb{N}, \_) : \mathbf{Graph}_*^\sharp \to \mathbf{Set}$ factors through the category **UMet**.

Proof. The only thing to check is that $f_* : (\mathbb{N}, G) \to (\mathbb{N}, G')$ is non-expanding, for any $f : G \rightharpoonup G'$. Take any $\vec{u}, \vec{v} \in (\mathbb{N}, G)$. Suppose that $d(\vec{u}, \vec{v}) = 2^{-n}$. Hence, there are vertices $w_0, \ldots, w_n$ of G such that $\vec{u} = (w_0 \to \ldots \to w_n \to u_{n+1} \to \ldots)$ and $\vec{v} = (w_0 \to \ldots \to w_n \to v_{n+1} \to \ldots)$. Applying f_* to $\vec{u}$ and $\vec{v}$, we obtain that $f_*\vec{u}$ and $f_*\vec{v}$ share a common head of length at least n, i.e., $d(f_*\vec{u}, f_*\vec{v}) \leq 2^{-n} = d(\vec{u}, \vec{v})$ and f_* is non-expanding. □

The above proposition allows us to consider $(\mathbb{N}, \_) : \mathbf{Graph}^\sharp_* \to \mathbf{Set}$ as a functor $(\mathbb{N}, \_) : \mathbf{Graph}^\sharp_* \to \mathbf{UMet}$ instead. We introduce some additional concepts for reasoning about program behaviour, including executions, properties, and safety, which we can interpret within the framework we defined here. Infinite lists of elements of set S are called *executions* [1]. They represent sequences of states of some process, for whom S serve as a set of all possible states. The subset $P \subseteq \langle\!\langle S \rangle\!\rangle$ is identified with its characteristic function $\chi_P : \langle\!\langle S \rangle\!\rangle \to \mathsf{Bool}$, and the latter is called a *property*.

A property P is said to be a *safety property*, if P does not hold for an execution $\vec{s}$, then at some state s_i some "*bad thing*" must happen. Such a "bad thing" must be irremediable because a safety property states that the "bad thing" never happens during execution. In other words, we characterize $P \subseteq \langle\!\langle S \rangle\!\rangle$ via a statement about its complement:

$$\vec{s} \notin P \;\; \Leftrightarrow \;\; (\exists\, n \geq 0)(\forall \vec{t} \in \langle\!\langle S \rangle\!\rangle)\{(s_0, \ldots, s_n, t_0, t_1, \ldots) \notin P\}$$

Using the ultrametric d for $\langle\!\langle S \rangle\!\rangle$, $\vec{s} \notin P$ is equivalent to $(\exists\, n \geq 0)\{B(\vec{s}, 2^n) \cap P = \varnothing\}$. In other words, the complement of P contains every point with some its neighborhood. So the safety property P uniquely defines a closed subset in $(\langle\!\langle S \rangle\!\rangle, d)$ and vice versa.

Another important class of properties of execution traces is *liveness* [1], which, like safety, can also be defined in topological terms. An arbitrary property of traces can be defined in terms of a liveness and safety property. The structured contract formalism, as well as blockchains in general, however, are more amenable to being studied from the point of view of guaranteeing safety. Investigating liveness in this setting is equally important, but poses more of a challenge. We, therefore, leave it for future work.

Recall also that we defined the trace $\mathsf{Trc}(f)$ of $f : (G; \mathring{V}) \rightharpoonup (G'; \mathring{V}')$ as the image of f_*, i.e., a subset of $(\mathbb{N}, G')$. Recall also that the representable functor $(\mathbb{N}, \_)$ factors through **UMet**, inducing an ultrametric in $(\mathbb{N}, G')$, so that traces of morphisms in $\mathbf{Graph}^\sharp_*$ are themselves ultrametric spaces. That is, we endow the subset $\mathrm{Im}(f_*) \subseteq (\mathbb{N}, G')$ with the ultrametric structure induced via metric from $(\mathbb{N}, G')$.

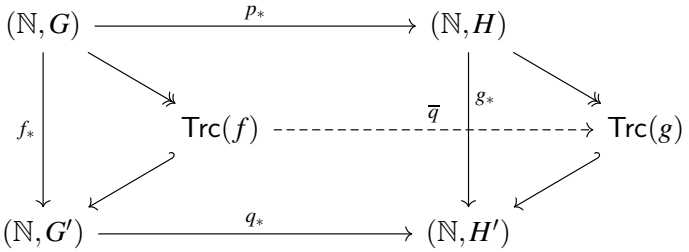
We obtain that a map in $\mathbf{Graph}^\sharp_*$ bewteen two graphs induces a morphism between sets of infinite paths in the those graphs in a way that preserves the ultrametric structure as it is non-expanding:

Lemma 4.4.5. A commutative square

$$\begin{array}{ccc} (G; \mathring{V}) & \xrightarrow{\;p\;} & (H; \mathring{W}) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle g} \\ (G'; \mathring{V}') & \xrightarrow{\;q\;} & (H'; \mathring{W}') \end{array}$$

in $\mathbf{Graph}^\sharp_*$ induces a non-expanding postcomposition function $\overline{q} : \mathsf{Trc}(f) \to \mathsf{Trc}(g)$.

Proof. Applying the functor $(\mathbb{N}, \_) : \mathbf{Graph}^\sharp_* \to \mathbf{UMet}$ to a given square, we obtain a commutative diagram of non-expanding maps



For any $\vec{s} \in \mathsf{Trc} f = \mathrm{Im}\, f_*$ there is $\vec{v} \in (\mathbb{N}, G)$ such that $f_*(\vec{v}) = \vec{s}$. Since $q_* f_* = g_* p_*$, then $\overline{q}(\vec{s}) = q_*(\vec{s}) = q_*(f_*(\vec{v})) = g_*(p_*(\vec{v})) \in \mathrm{Im}\, g_*$. Hence, we checked that $\overline{q}$ is well-defined, i.e., its codomain is $\mathsf{Trc}(g)$. Since the inclusion of a subspace $\mathrm{Im}(f_*)$ in $(\mathbb{N}, G')$ is a non-expanding map, so is $\overline{q}$ as composition of the inclusion and q_*. □

Applying the above result to structured contracts, we get:

Corollary 4.4.6 (Trace-mapping lemma). A structured contract $(\mathsf{STRUC}, \pi, \kappa)$ for LEDGER induces a non-expanding map

$$\overline{\pi} : \mathsf{Trc}(\mathsf{LEDGER}) \to \mathsf{Trc}(\mathsf{STRUC})$$

between ultrametric spaces.

The trace-mapping lemma expresses a practical result of this work: a safety property (i.e., a closed subset of valid traces) of $\mathsf{Trc}(\mathsf{STRUC})$ necessarily has a corresponding safety property in $\mathsf{Trc}(\mathsf{LEDGER})$, i.e., a closed subset of in the domain of the non-expanding map induced by $(\mathsf{STRUC}, \pi, \kappa)$. In particular, $\mathsf{Trc}(\mathsf{STRUC})$, which is also a closed subset of itself, has a closed preimage in $\mathsf{Trc}(\mathsf{LEDGER})$. This preimage is a safety property of $\mathsf{Trc}(\mathsf{LEDGER})$. This safety property can be expressed in the "a specific bad thing cannot be fixed if it happens" language as: if a trace $\vec{v} \in \mathsf{Trc}(\mathsf{LEDGER})$ is not in the preimage of $\mathsf{Trc}(\mathsf{STRUC})$, there is necessarily some i, such that $v_0 \to \ldots \to v_i$ is a prefix of $\vec{v}$, and no trace of the form $v_0 \to \ldots \to v_i \to u_{i+1} \to \ldots$ maps to a trace in $\mathsf{Trc}(\mathsf{STRUC})$.

Preimages of traces also carry important information about them. For this reason, we define the concept of a *full lift* of a trace:

Definition 4.4.7. Each element $\vec{s}$ of $\mathsf{Trc}(f) = \mathrm{Im}(f_*)$ has *full lift* $\vec{v} \in (\mathbb{N}, G)$ via $f_*(\vec{v}) = \vec{s}$.

In other words, for every trace $\vec{s}$ of f, which is an infinite path, there is an infinite path $\vec{v}$ that is mapped to it. The latter yields that each head $\vec{s}[n]$ of $\vec{s}$ has a lift $\vec{v}[n]$ via $f_*(\vec{v}[n]) = \vec{s}[n]$. In the following proposition, we express when the converse holds,

Proposition 4.4.8. Let $f : G \rightharpoonup G'$ be a morphism in $\mathbf{Graph}^{\natural}_*$. Then $\overline{\mathsf{Trc}(f)} = \bigcap_{n \geq 0} \mathsf{Trc}_n(f)$, where $\overline{\mathsf{Trc}(f)}$ is the closure of $\mathsf{Trc}(f)$ in $(\mathbb{N}, G')$ and $\mathsf{Trc}_n(f) := \{\vec{s} : \mathbb{N} \to G' \mid \exists\, \vec{v} : \mathbb{N} \to G :\ f_*(\vec{v}[n]) = \vec{s}[n]\}$ is a closed set of infinite paths in G' that has *n-truncated lifts* in $(\mathbb{N}, G)$. Hence, $\mathsf{Trc}(f)$ is closed if and only if the following holds: any infinite path in G' has a full lift in G if and only if an infinite path in G' has an n-truncated lift in G for all $n \geq 0$.

Proof. Let $\vec{s}$ be a limit point of $\mathsf{Trc}_n(f)$. Then $\exists\, \vec{t} \in B(\vec{s}, 2^{-n}) \cap \mathsf{Trc}_n(f) \neq \varnothing$. Hence, $\vec{t} = (s_0 \to \ldots \to s_n \to t_{n+1} \to t_{n+2} \to \ldots)$ is an infinite path in G' and there is $\vec{v} \in (\mathbb{N}, G)$ such that $f_*(\vec{v}[n]) = \vec{t}[n]$ i.e., $t_i = s_i = f(v_i)$ for $0 \leq i \leq n$. Therefore, $f_*(\vec{v}[n]) = \vec{s}[n]$ so $\vec{s} \in \mathsf{Trc}_n(f)$ and $\mathsf{Trc}_n(f)$ is closed. Obviously, $\mathsf{Trc}(f) \subseteq \mathsf{Trc}_n(f)$ for every $n \geq 0$, so $\mathsf{Trc}(f)$ is a subset of $\bigcap_{n \geq 0} \mathsf{Trc}_n(f)$, which is a closed set. Hence, $\overline{\mathsf{Trc}(f)} \subseteq \bigcap_{n \geq 0} \mathsf{Trc}_n(f)$ as the closure is the smallest closed set containing $\mathsf{Trc}(f)$.

Conversely, if $\vec{s} \in \mathsf{Trc}_n(f)$ for all $n \geq 0$, then $(\forall\, n \geq 0)(\exists\, \vec{v} \in (\mathbb{N}, G))\{f_*(\vec{v}[n]) = \vec{s}[n]\}$. The latter means that $d(f_*(\vec{v}), \vec{s}) < 2^{-n}$ for each $n \geq 0$. But $f_*(\vec{v}) \in \mathrm{Im}(f_*)$, so $B(\vec{s}, 2^{-n}) \cap \mathsf{Trc}(f) \neq \varnothing$ for each $n \geq 0$. Therefore, $\vec{s}$ is a limit point of $\mathsf{Trc}(f)$ and $\bigcap_{n \geq 0} \mathsf{Trc}_n(f) \subseteq \overline{\mathsf{Trc}(f)}$ which was desired. □

5 Properties of LEDGER system

Since we do not give implementation details or concrete examples of structured contracts, presenting examples of their properties is left for future work. In this section, we focus on properties of the LEDGER system itself, for which we first introduce additional notation and terminology.

The set UTxO is assumed to be "*well-founded*": for any $u \in \mathsf{UTxO}_0$ and any key-value pair $((b,n),o) \in u$, there is a transaction $t \in T$ such that $b = h(t)$, $\mathsf{inputs}(t) = \varnothing$ i.e. hash part of any key in a valid initial UTxO state $u \in \mathsf{UTxO}_0$ comes from a transaction with empty list of inputs. In the rest of the paper we'll assume that UTxO is well-founded.

An updated state $u' = (u \setminus \mathsf{getORefs}(t)) \sqcup \mathsf{mkOuts}(t)$ is constructed from a triple (q,u,t) satisfying checkTx via the functions getORefs and mkOuts. The condition checkTx guarantees that all triples (b,n,o) extracted from $\mathsf{inputs}(t)$ belong to u. After subtracting $\mathsf{getORefs}(t)$ from u, the function mkOuts joins new pairs $\{(h(t),n,o) \mid (n,o) \in \mathsf{outs}(t)\}$ to the result from the previous step. For a lift $(q_0,u_0,t_0) \to (q_1,u_1,t_1) \to \ldots$ of a valid LEDGER trace $u_0 \to u_1 \to \ldots u_k \to \ldots$ we introduce the following notations: $r_i := \mathsf{getORefs}(t_i)$, $c_i := \mathsf{mkOuts}(t_i)$. Therefore, the sequence of valid LEDGER states $u_{k+1} = (u_k \setminus r_k) \cup c_k$, for $k \geq 0$, where $u_0 \in \mathsf{UTxO}_0$.

5.1 Replay and trivial update protection

An important property of UTxO ledgers is that an attacker is not able to disrupt the operation of the ledger program by re-applying an existing transaction. A related property is that it is not possible to apply a transaction that does not change the ledger state. We can formalize these as safety properties of ledger traces in the following way:

Theorem 5.1.1 ((Replay and trivial update protection)**.** Given an infinite path $(q_0,u_0,t_0) \to (q_1,u_1,t_1) \to \ldots$ in a graph corresponding to LEDGER, for any indexes $i < j$

(a) $t_i \neq t_j$ (replay protection);

(b) $u_i \neq u_j$ (trivial update protection).

Both (a) and (b) are safety properties of $\mathsf{Trc}(\mathsf{LEDGER})$.

Proof. (a) Suppose $i < j$ is the minimal pair such that $t_i = t_j$. Then $r_i \subseteq u_i = (u_{i-1} \setminus r_{i-1}) \cup c_{i-1} \subseteq u_{i-1} \cup c_{i-1} \subseteq u_0 \cup c_0 \cup c_1 \cup \ldots \cup c_{i-1}$. A similar chain of inclusions yields $r_j \subseteq u_j \subseteq (u_i \setminus r_i) \cup c_i \cup \ldots \cup c_{j-1}$. Since $r_i = r_j$, the set r_j has no common elements with $u_i \setminus r_i$, so $r_j \subseteq c_i \cup \ldots \cup c_{j-1}$ and $r_i \cap r_j \subseteq (u_0 \cup c_0 \cup \ldots \cup c_{i-1}) \cap (c_i \cup \ldots \cup c_{j-1})$.

Claim 1. For any $l \geq 0 : u_0 \cap c_l = \varnothing$. Suppose $(b,n,o) \in u_0 \cap c_l$, for some $l \geq 0$. Then b is a hash of a transaction $\mathring{t}$ with an empty input, by the well-foundedness of UTxO. On the other hand, b is a hash of a transaction t_l with a non-empty input, since (q_l,u_l,t_l) satisfies checkTx. Hence, $\mathring{t} = t_l$, by the injectivity of h, which is a contradiction, as their inputs differ. So, $u_0 \cap c_l = \varnothing$.

Claim 2. For any $0 \leq l \leq i-1$ and $i \leq m \leq j-1 : c_l \cap c_m = \varnothing$. Hash components of all elements in c_l and c_m are $h(t_l)$ and $h(t_m)$, respectively. If these sets share an element, hash of this element is $h(t_l) = h(t_m)$. From the injectivity of h follows that $t_l = t_m$. The latter equality contradicts the assumption about the minimality of the pair (i,j). Therefore, $c_l \cap c_m = \varnothing$.

Claims 1 and 2 imply $r_i \cap r_j = \varnothing$, which means that $r_i = r_j = \varnothing$. Hence, $\mathsf{inputs}(t_i) = \mathsf{inputs}(t_j) = \varnothing$, that contradicts the fact that (q_i,u_i,t_i) and (q_j,u_j,t_j) satisfy checkTx.

(b) Suppose there is a pair $i < j$ such that $u_i = u_j$. An equality of sets $u_j = (u_{j-1} \setminus r_{j-1}) \cup c_{j-1}$ imply that $c_{j-1} \subseteq u_i \subseteq u_{i-1} \cup c_{i-1} \subseteq u_{i-2} \cup c_{i-2} \cup c_{i-1} \subseteq \ldots u_0 \cup c_0 \cup \ldots \cup c_{i-1}$ By Claim 1 $u_0 \cap c_{j-1} = \varnothing$.

If $c_j \cap c_l \neq \varnothing$, where $0 \leq l \leq i-1$, then $t_j = t_l$, which is impossible by part (a). Hence, $(u_0 \cup c_0 \cup \ldots \cup c_{i-1}) \cap c_{j-1} = \varnothing$ and $u_i \neq u_j$.

Claim 3. Because the full space (which is always closed) satisfies this property, this property represents a closed set, and is therefore a safety property. □

An important corollary of the above theorem is that sets we add to or delete from a given ledger state (UTxO set) are pairwise disjoint. This will be used in an upcoming result.

Corollary 5.1.2. If $h : \mathsf{Tx} \to \mathsf{ByteString}$ is injective, then: (i) sets $u_0,\ c_0,\ c_1,\ \ldots$ are pairwise disjoint; (ii) sets $r_0, r_1, \ldots$ are pairwise disjoint. Moreover, $u_{k+1} = (u_k \setminus r_k) \sqcup c_k$ for any $k \leq 0$.

We note here that the above properties would also be safety properties when considered as properties of arbitrary (not necessarily valid) traces. This can be justified as follows: any trace containing a prefix such that for $i \neq j$, transactions $t_i = t_j$ will never satisfy the replay protection property, regardless of its suffix. The "bad thing" cannot be fixed. Similarly, a trace containing a trivial update at states $u_i = u_j$ cannot be "fixed" by any suffix.

5.2 UTxO transaction commutativity

The UTxO set in the LEDGER transition system enjoys the transaction commutativity property meaning that *the order of applying transactions to a valid initial state is irrelevant and always returns the same result.* Still we should keep in mind that every single transaction application must be validated by checkTx. While this property has to do with finite sequences of states and transactions, we can express it a property of (infinite) execution traces.

Theorem 5.2.1 (UTxO transaction commutativity)**.** Let u_0 is a well-founded UTxO state. Suppose we have two traces with (possibly) distinct length-$n+1$ prefixes, and an arbitrary suffix,

$$(q_0, u_0, t_0) \longrightarrow (q_1, u_1, t_1) \longrightarrow \ldots \longrightarrow (q_n, u_n, t_n) \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \ldots$$

$$(q'_0, u'_0, t'_0) \longrightarrow (q'_1, u'_1, t'_1) \longrightarrow \ldots \longrightarrow (q'_n, u'_n, t'_n) \longrightarrow s'_1 \longrightarrow s'_2 \longrightarrow \ldots$$

in the simple graph corresponding to LEDGER, where $u_0 = u'_0 \in \mathsf{UTxO}_0$ and $(t'_0, \ldots, t'_n)$ is a permutation of $(t_0, \ldots, t_n)$. If h is injective, then $u_n = u'_n$.

Before we prove the above result, we motivate the proof and illustrate this property with an example:

Example 5.2.2. Let $(t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7)$ and $(t_3, t_1, t_6, t_2, t_5, t_7, t_0, t_4)$ be valid sequences of transactions that are applied to a valid initial state $u_0 \in \mathsf{UTxO}_0$. Then

$$\begin{array}{lcll|lcll}
u_1 & = & (u_0 \setminus r_0) \sqcup c_0 & \subseteq\ u_0 \sqcup c_0 & u'_1 & = & (u_0 \setminus r_3) \sqcup c_3 & \subseteq\ u_0 \sqcup c_3 \\
u_2 & = & (u_1 \setminus r_1) \sqcup c_1 & \subseteq\ u_0 \sqcup c_{01} & u'_2 & = & (u'_1 \setminus r_1) \sqcup c_1 & \subseteq\ u_0 \sqcup c_{13} \\
u_3 & = & (u_2 \setminus r_2) \sqcup c_2 & \subseteq\ u_0 \sqcup c_{012} & u'_3 & = & (u'_2 \setminus r_6) \sqcup c_6 & \subseteq\ u_0 \sqcup c_{136} \\
u_4 & = & (u_3 \setminus r_3) \sqcup c_3 & \subseteq\ u_0 \sqcup c_{0123} & u'_4 & = & (u'_3 \setminus r_2) \sqcup c_2 & \subseteq\ u_0 \sqcup c_{1236} \\
u_5 & = & (u_4 \setminus r_4) \sqcup c_4 & \subseteq\ u_0 \sqcup c_{01234} & u'_5 & = & (u'_4 \setminus r_5) \sqcup c_5 & \subseteq\ u_0 \sqcup c_{12356} \\
u_6 & = & (u_5 \setminus r_5) \sqcup c_5 & \subseteq\ u_0 \sqcup c_{012345} & u'_6 & = & (u'_5 \setminus r_7) \sqcup c_7 & \subseteq\ u_0 \sqcup c_{123567} \\
u_7 & = & (u_6 \setminus r_6) \sqcup c_6 & \subseteq\ u_0 \sqcup c_{0123456} & u'_7 & = & (u'_6 \setminus r_0) \sqcup c_0 & \subseteq\ u_0 \sqcup c_{0123567} \\
u_8 & = & (u_7 \setminus r_7) \sqcup c_7 & \subseteq\ u_0 \sqcup c_{01234567} & u'_8 & = & (u'_7 \setminus r_4) \sqcup c_4 & \subseteq\ u_0 \sqcup c_{01234567}
\end{array}$$

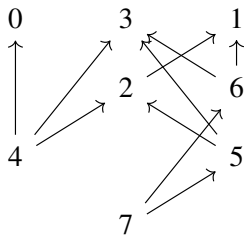
where $c_{i_1 i_2 \ldots i_k} = \cup_{j=1}^{k} c_{i_j}$. Since $c_A \cap c_B = c_{A \cap B}$ for any subsets A, B of $\{0, \ldots, 7\}$,

$$
\begin{array}{lcll}
r_0 & \subseteq & u_0 \cap (u_0 \sqcup c_{123567}) & = & u_0 \\
r_1 & \subseteq & (u_0 \sqcup c_0) \cap (u_0 \sqcup c_3) & = & u_0 \\
r_2 & \subseteq & (u_0 \sqcup c_{01}) \cap (u_0 \sqcup c_{136}) & = & u_0 \sqcup c_1 \\
r_3 & \subseteq & (u_0 \sqcup c_{012}) \cap u_0 & = & u_0 \\
r_4 & \subseteq & (u_0 \sqcup c_{0123}) \cap (u_0 \sqcup c_{0123567}) & = & u_0 \sqcup c_{0123} \\
r_5 & \subseteq & (u_0 \sqcup c_{01234}) \cap (u_0 \sqcup c_{1236}) & = & u_0 \sqcup c_{123} \\
r_6 & \subseteq & (u_0 \sqcup c_{012345}) \cap (u_0 \sqcup c_{13}) & = & u_0 \sqcup c_{13} \\
r_6 & \subseteq & (u_0 \sqcup c_{012345}) \cap (u_0 \sqcup c_{13}) & = & u_0 \sqcup c_{13} \\
r_7 & \subseteq & (u_0 \sqcup c_{0123456}) \cap (u_0 \sqcup c_{12356}) & = & u_0 \sqcup c_{12356}
\end{array}
$$

The above inclusions yield:

(0) subsets r_0, r_1, r_3 are subtracted from u_0 in any order, then c_0, c_1, c_3 are attached;
(1) subsets r_2, r_6 are subtracted from the result of (0) in any order, then c_2, c_6 are attached;
(2) subsets r_4, r_5 are subtracted from the result of (1) in any order, then c_4, c_5 are attached;
(3) subset r_7 is subtracted from the result of (2), then c_7 is attached.

Hence, the set $\{0,\dots,7\}$ is equipped with a partial order structure: $i < j$ if the subtraction of r_i can potentially depend on attaching of c_j. In the case of our example, the Hasse diagram of a poset is



Vertices at the top of the diagram (sinks) are *level 0* vertices. A level of a vertex i is a length of a maximal path from i to vertices of level 0. Hence, we obtain a level partition of the set of vertices: (0) level 0 vertices: 0,1,3; (1) level 1 vertices: 2,6; (2) level 2 vertices: 4,5; (3) level 3 vertex: 7.

From our construction follows that transactions labeled by vertices of the same level commute, and a pair (t_i, t_j) can be swapped to (t_j, t_i), if the level of i is less than the level of j.

Fixing the ordering between the initial set of transactions, we obtain a canonical presentation of the set of transactions $(0,1,3 \mid 2,6 \mid 4,5 \mid 7)$ and the canonical path, where $w_0 = u_0$:

$$(w_0,t_0) \to (w_1,t_1) \to (w_2,t_3) \to (w_3,t_2) \to (w_4,t_6) \to (w_5,t_4) \to (w_6,t_5) \to (w_7,t_7)$$

Finally, we present a way how to transform one valid transaction sequence into another. Indeces of numbers represent their levels, adjacent red numbers are swapped in the next row.

$$
\begin{array}{cccccccc}
(0_0 & 1_0 & 2_1 & 3_0 & 4_2 & 5_2 & 6_1 & 7_3) \\
(0_0 & 1_0 & 3_0 & 2_1 & 4_2 & 6_1 & 5_2 & 7_3) \\
(0_0 & 1_0 & 3_0 & 2_1 & 6_1 & 4_2 & 5_2 & 7_3)
\end{array}
\qquad
\begin{array}{cccccccc}
(3_0 & 1_0 & 6_1 & 2_1 & 5_2 & 7_3 & 0_0 & 4_2) \\
(1_0 & 3_0 & 6_1 & 2_1 & 5_2 & 0_0 & 7_3 & 4_2) \\
(1_0 & 3_0 & 2_1 & 6_1 & 5_2 & 0_0 & 4_2 & 7_3) \\
(1_0 & 3_0 & 2_1 & 6_1 & 0_0 & 5_2 & 4_2 & 7_3) \\
(1_0 & 3_0 & 2_1 & 0_0 & 6_1 & 5_2 & 4_2 & 7_3) \\
(1_0 & 3_0 & 0_0 & 2_1 & 6_1 & 4_2 & 5_2 & 7_3) \\
(1_0 & 0_0 & 3_0 & 2_1 & 6_1 & 4_2 & 5_2 & 7_3) \\
(0_0 & 1_0 & 3_0 & 2_1 & 6_1 & 4_2 & 5_2 & 7_3)
\end{array}
$$

The more valid permutations of the set of transactions we have, more precisely we'll define the canonical form. We can distill the approach taken in the example into a proof:

Proof. Let

$$(q_0,u_0,t_0) \longrightarrow (q_1,u_1,t_1) \longrightarrow \ldots \longrightarrow (q_n,u_n,t_n) \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \ldots$$

$$(q'_0,u'_0,t'_0) \longrightarrow (q'_1,u'_1,t'_1) \longrightarrow \ldots \longrightarrow (q'_n,u'_n,t'_n) \longrightarrow s'_1 \longrightarrow s'_2 \longrightarrow \ldots$$

be two traces in the simple graph corresponding to LEDGER, where $u_0 = u'_0 \in \mathsf{UTxO}_0$ and $(t'_0,\ldots,t'_n)$ is a permutation of $(t_0,\ldots,t_n)$. Let us define sets $r_i = \mathsf{getORefs}(t_i)$ and $c_i = \mathsf{mkOuts}(t_i)$ for $0 \leq i \leq n$. Also, let $u_\cup = u_0 \cup_{0\leq i\leq n} c_i$ be the result of adding all outputs from all transactions $t_0,...,t_n$ to u_0, which is also the same as adding all outputs from all transactions $t'_0,...,t'_n$ to u_0. The final states u_n and u'_n, as well as all intermediate states, are contained in $u_\cup$, since the only other operation involved in updating the state is removing UTxO entries.

Set $r = \cup_{0\leq i\leq n} r_i$ consists of UTxO entries that $t_0,...,t_n$ (or $t'_0,...,t'_n$) remove from $u_\cup$. This r is such that $r \subseteq u_\cup$, which is guaranteed by checkTx. To show that both $u_n = u_\cup \setminus r = u'_n$, we proceed by contradiction. Suppose $a \notin u_n$, but $a \in u'_n$. By above, $a \in u_\cup$. Now, a must have been removed by some t_i from $u_\cup$ (and therefore, also some t'_j). Since a is in u'_n, it must have been re-added by another t_k. From the replay protection property, we get a contradiction: t_k must have the same encoding as a previous transaction that added a, which contradicts Corollary 5.1.2. By similar logic, we can show that any $a \in u_n$ must also be in u'_n. □

The following algorithm produces a canonical form of a transaction list:

Algorithm 5.2.3 (Canonical form of transaction sequence)**. Input data**. A finite path $\{(q_i,u_i,t_i)\}_{i=0}^n$ in a graph corresponding to LEDGER, where $u_0 \in \mathsf{UTxO}_0$.

Output data. A set of all possible finite paths $\{(q'_i,u'_i,t'_i)\}_{j=0}^n$, where $u'_0 = u_0$ and $(t'_0,\ldots,t'_n)$ is a permutation of $(t_0,\ldots,t_n)$.

Step 1. For every i determine the set $K_i := \left\{ j \mid r_i \cap c_j \neq \varnothing \right\}$.

Step 2. Define a poset structure on $\{0,\ldots,n\} : i < j$ if and only if $j \in K_i$.

Step 3. Say that an index i has level 0, if $K_i = \varnothing$. If $K_i \neq \varnothing$, say that a level of i is a maximum of path lengths from i to indexes of level 0 in Hasse diagram of the poset on $\{0,\ldots,n\}$.

Step 4. Make a total order on the set $\{0,\ldots,n\} : i < j$ if and only if the level of i is less that the level of j, or $i < j$ in the natural ordering, if i and j have the same level. Call a sequence representing this total order $(t_{i_0},\ldots,t_{i_n})$ the *canonical presentation* of $(t_1,\ldots,t_n)$.

Step 5. An *elementary swap* of $(t_{j_0},\ldots,t_{j_k},\ldots,t_{j_l},\ldots,t_{j_n})$ is a sequence $(t_{j_0},\ldots,t_{j_l},\ldots,t_{j_k},\ldots,t_{j_n})$ if $l < k$ in a total order. Return as an output a set of all sequences $(t_{j_0},\ldots,t_{j_n})$ that can be obtained from the canonical presentation by a finite number of elementary swaps.

Note that the decision procedure has the input of a single transaction list, while the transaction commutativity theorem is formulated in terms of two lists that are permutations of each other. It is more natural to express this theorem in a way that is symmetric with respect to the two permutations of transaction lists. To state it as a property, we can rephrase it as a property of a single trace: the property holds for a given trace with the required structure whenever, given any other trace with the required prefix and suffix, the final state of the prefix must be the same for both traces.

Like in the case of replay protection as well as trivial update protection, this is a safety property by virtue of being true for the entire space. This is also a safety property when considered as a property of all possible UTxO state traces (not just the valid ones). Suppose s is a trace of UTxO states that has a n-length prefix that is not generated by applying a valid lists of transactions, and an arbitrary suffix. If another trace s' exists such that s and s' violate the transaction commutativity property, changing the suffix starting at $n+1$ position in the trace of s will never "fix" s. So, when the transaction commutativity property breaks in a finite prefix, it cannot be remedied, as required by the definition of a safety property.

6 Conclusion

6.1 Related Work

In this work, we use tools from different areas of mathematics to study program (i.e., smart contract) executions on the ledger. The approach to representing ledger semantics is used in existing work in UTxO ledger and contract formalization [9] [19] [30]. Transition systems are commonly represented by graphs, with edges corresponding to possible state transitions [22]. However, the graph generated directly by the small-steps semantics of ledgers (or other programs) does not contain edges for multi-step transitions, and it also does not exclude "bad" starting states. For this reason, we instead consider possible paths in the resulting graphs, and limit our attention to only paths with certain "good" starting states. Possible paths in our valid transition graphs align with the notion of execution traces [1], however, they are generated according to the small-steps specifications, and are therefore not arbitrary.

The definition of ledger implementation relation, which we build on this work, is a reversal of the the classic *simulation* relation [22]. We introduce this terminology in this way because it is descriptive of the unique architecture of programs running on the UTxO ledger, which is different from the traditional communicating automata-style distributed programs. This is because stateful UTxO programs are implemented using multiple permission-like pieces of code that control what aggregates (or pieces) of the ledger state that transactions are permitted to control. It also does not make sense to model such programs via a notion that is used in studying transition systems — subsystems [27]. This is because a subsystem relates the evolution of a subset of states to the evolution of the entire system, which is not what we study here.

Rather than subsystems, we use sieve-defined homomorphisms as the formalism that relates the evolution of the implemented program state to the that of the ledger state. In category theory, a sieve is a is a generalization of the notion of ideals, which guarantees that arrows starting in a given set of objects also end in that same set [14]. Now, categories are closed under arrow composition, and our graphs are not closed under edge composition. We, however, apply the defining property of categorical sieves to graph homomorphisms.

Algebraic descriptions of UTxO transaction processing appear in existing work [13]. However, no models of implementations of programs on the ledger are described within this model. The ledger structure itself has been modelled categorically, using monoidal functors to represent the structure of UTxO set updates. It may be possible to combine this model and ours in future work.

Certain safety properties have been defined, and are consistently being checked in production ledger systems as part of property testing frameworks. This involves generating arbitrary transactions, and applying them to generated states to verify that the property is not violated [3]. In the future, we also plan to incorporate the results in this work into more realistic models.

6.2 Discussion

Programs for UTxO-style ledgers, which are composed entirely out of predicates on state updates, differs significantly from common programming paradigms. For this reason, a specialized model is required to describe it in a principled way. The structured contract framework provides a rigorous and principled way to establish a relation between the contents of pieces of user-defined code, including a small-steps specification, an implementation, and a proof that a single valid ledger step corresponds to a valid program step. However, the capacity for reasoning within this framework is very limited without additional

[3] `https://github.com/IntersectMBO/cardano-ledger`

structure. In this work, we define the required structure.

Our contribution includes a rigorous definition of a valid program trace for a given small-step semantics and set of valid initial states. Next, we construct a category that we use to model the relation between the behavior of the ledger and a program implemented on it. This category has simple graphs as objects, and partial sieve-defined homomorphisms as morphisms. We define maps between valid traces, which are paths in the graph, in terms of maps in $\mathbf{Graph}^{\sharp}_{*}$. We then show how such maps are induced by a structured contract. We apply an existing metric defined on arbitrary execution traces (i.e., paths in a complete directed state graph) to paths in our graph, which correspond to valid traces only. We show that the metric we applied is, in fact, an ultrametric. This allows us to demonstrate that the maps in our category are non-expanding and continuous, and therefore, for any safety property of a program, its ledger implementation has an associated safety property.

We go on to prove certain important safety properties of the ledger, including commutativity, replay protection, and trivial update protection. These properties all rely on the ability to reason about the prefix of a given trace, which was previously not supported in the structured contract framework. Assuming certain consequences of these properties is often required in practice for proving correctness of program implementations on the ledger [29].

We formalized the notion of safety in the UTxO ledger programming context, leaving liveness for future work. We intend to use our model to study relationships between structured contracts on a single ledger, as well as the possibility of composing them. We are also interested in investigating unique quirks of smart contracts in the EUTxO model, such as the double satisfaction problem [29], and formalize what it means for a transaction's changes to the ledger to be predictable (building on the transaction commutativity property). Another interesting direction of research would be to apply similar methods and techniques we have defined here to other blockchains that have formal models, such as Algorand [3] or Bitcoin [26].

References

[1] Bowen Alpern & Fred B. Schneider (1985): *Defining liveness*. *Information Processing Letters* 21(4), pp. 181–185, doi:10.1016/0020-0190(85)90056-0.

[2] Christel Baier & Joost-Pieter Katoen (2008): *Principles of model checking*. MIT Press.

[3] Massimo Bartoletti, Andrea Bracciali, Cristian Lepore, Alceste Scalas & Roberto Zunino (2021): *A formal model of Algorand smart contracts*, doi:10.48550/arXiv.2009.12140. arXiv:2009.12140.

[4] Francis Borceux (1994): *Handbook of categorical algebra: volume 1, Basic category theory*. 1, Cambridge University Press, doi:10.1017/CBO9780511525865.

[5] Francis Borceux (1994): *Handbook of Categorical Algebra: Volume 3, Sheaf Theory*. 3, Cambridge university press, doi:10.1017/CBO9780511525865.

[6] Carolyn Brown & Doug Gurr (1993): *Temporal logic and categories of Petri nets*. In Andrzej Lingas, Rolf Karlsson & Svante Carlsson, editors: *20th International Colloquium on Automata, Languages, and Programming (ICALP 93)*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 570–581, doi:10.1007/3-540-56939-1_103.

[7] Vitalik Buterin (2014): *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. https://ethereum.org/en/whitepaper/.

[8] Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Jann Müller, Michael Peyton Jones, Polina Vinogradova & Philip Wadler (2020): *Native Custom Tokens in the Extended UTXO Model*. In Tiziana Margaria & Bernhard Steffen, editors: *Leveraging Applications of Formal Methods, Verification and Validation: Applications - 9th International Symposium on Leveraging Applications*

of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part III, Springer, pp. 89–111, doi:10.1007/978-3-030-61467-6_7.

[9] Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones & Philip Wadler (2020): *The Extended UTXO Model*. In Matthew Bernhard, Andrea Bracciali, L. Jean Camp, Shin'ichiro Matsuo, Alana Maurushat, Peter B. Rønne & Massimiliano Sala, editors: *Financial Cryptography and Data Security*, Springer International Publishing, Cham, pp. 525–539, doi:10.1007/978-3-030-54455-3_37.

[10] Jared Corduan, Matthias Güdemann & Polina Vinogradova (2019): *A Formal Specification of the Cardano Ledger*. https://github.com/input-output-hk/cardano-ledger/releases/latest/download/shelley-ledger.pdf.

[11] Volker Diekert & Yves Métivier (1997): *Partial Commutation and Traces*, pp. 457–533. 3, Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-59126-6_8.

[12] Ergo Team (2019): *Ergo: A Resilient Platform For Contractual Money*. https://ergoplatform.org/docs/whitepaper.pdf.

[13] Murdoch J. Gabbay (2021): *Algebras of UTxO blockchains*. *Mathematical Structures in Computer Science* 31(9), p. 1034–1089, doi:10.1017/S0960129521000438.

[14] Jean Giraud (1962-1964): *Analysis situs*. *Séminaire Bourbaki* 8, pp. 189–199. Available at http://eudml.org/doc/109657.

[15] LM Goodman (2014): *Tezos—a self-amending crypto-ledger White paper*. https://tezos.com/whitepaper.pdf.

[16] Heine Halberstam & Hans Egon Richert (2013): *Sieve methods*. Courier Corporation.

[17] Ulrich Hensel & David Spooner (1996): *A view on implementing processes: Categories of circuits*. In Magne Haveraaen, Olaf Owe & Ole-Johan Dahl, editors: *Recent Trends in Data Type Specification*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 237–254, doi:10.1007/3-540-61629-2_46.

[18] Andre Knispel, Orestis Melkonian, James Chapman, Alasdair Hill, Joosep Jääger, William DeMeo & Ulf Norell (2024): *Formal Specification of the Cardano Blockchain Ledger, Mechanized in Agda*. In Bruno Bernardo & Diego Marmsoler, editors: *5th International Workshop on Formal Methods for Blockchains (FMBC 2024)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 2:1–2:18, doi:10.4230/OASIcs.FMBC.2024.2.

[19] Andre Knispel & Polina Vinogradova (2021): *A Formal Specification of the Cardano Ledger integrating Plutus Core*. https://github.com/input-output-hk/cardano-ledger/releases/latest/download/alonzo-ledger.pdf.

[20] Marc Krasner (1944): *Nombres semi-réels et espaces ultramétriques*. *Comptes-Rendus de l'Académie des Sciences* 2, p. 219.

[21] Saunders Mac Lane (2013): *Categories for the working mathematician*. 5, Springer Science & Business Media, doi:10.1007/978-1-4757-4721-8.

[22] Robin Milner (1980): *A calculus of communicating systems*. Springer, doi:10.1007/3-540-10235-3.

[23] S. Nakamoto (2008): *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/en/bitcoin-paper.

[24] Chad Nester (2020): *A Foundation for Ledger Structures*. In Emmanuelle Anceaume, Christophe Bisière, Matthieu Bouvard, Quentin Bramas & Catherine Casamatta, editors: *2nd International Conference on Blockchain Economics, Security and Protocols, Tokenomics 2020, October 26-27, 2020, Toulouse, France*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 7:1–7:13, doi:10.4230/OASICS.TOKENOMICS.2020.7.

[25] Dusko Pavlovic & Douglas R. Smith (2003): *Software Development by Refinement*, pp. 267–286. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-540-40007-3_17.

[26] Kristijan Rupić, Lovro Rožić & Ante Derek (2020): *Mechanized Formal Model of Bitcoin's Blockchain Validation Procedures*. In Bruno Bernardo & Diego Marmsoler, editors: *2nd Workshop on Formal Methods for Blockchains (FMBC 2020)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 7:1–7:14, doi:10.4230/OASIcs.FMBC.2020.7.

[27] J. J.M.M. Rutten (1995): *A calculus of transition systems (towards universal coalgebra)*. https://ir.cwi.nl/pub/5060.

[28] The ZILLIQA Team (2017): *The ZILLIQA Technical Whitepaper*. https://docs.zilliqa.com/whitepaper.pdf.

[29] Polina Vinogradova & Orestis Melkonian (2025): *Message-Passing in the Extended UTxO Ledger*. In Jurlind Budurushi, Oksana Kulyk, Sarah Allen, Theo Diamandis, Ariah Klages-Mundt, Andrea Bracciali, Geoffrey Goodell & Shin'ichiro Matsuo, editors: *Financial Cryptography and Data Security. FC 2024 International Workshops*, Springer Nature Switzerland, Cham, pp. 150–169, doi:10.1007/978-3-031-69231-4_11.

[30] Polina Vinogradova, Orestis Melkonian, Philip Wadler, Manuel Chakravarty, Jacco Krijnen, Michael Peyton Jones, James Chapman & Tudor Ferariu (2024): *Structured Contracts in the EUTxO Ledger Model*. In Bruno Bernardo & Diego Marmsoler, editors: *5th International Workshop on Formal Methods for Blockchains (FMBC 2024)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 10:1–10:19, doi:10.4230/OASIcs.FMBC.2024.10.

[31] Jan Xie (2018): *Nervos CKB: A Common Knowledge Base for Crypto-Economy*. https://github.com/nervosnetwork/rfcs/blob/master/rfcs/0002-ckb/0002-ckb.md.

Query Answering in Lattice-based Description Logic

Krishna Manoorkar
School of Business and Economics
Vrije Universiteit Amsterdam*
Amsterdam, the Netherlands
k.b.manoorkar@vu.nl

Ruoding Wang
School of Business and Economics
Vrije Universiteit Amsterdam†
Amsterdam, the Netherlands
Department of Philosophy
Xiamen University
Xiamen, China
r.wang2@vu.nl

Recently, the description logic LE-$\mathscr{ALC}$ was introduced for reasoning in the semantic environment of the enriched formal contexts, and a tableaux algorithm was developed for checking the consistency of ABoxes in this logic [6, 7]. In this paper, we study the ontology-mediated query answering in LE-$\mathscr{ALC}$. In particular, we show that several different types of queries can be answered efficiently for LE-$\mathscr{ALC}$ knowledge bases with acyclic TBoxes using our tableaux algorithm directly or by extending it with some additional rules.

1 Introduction

Description logic (DL) [1] is a class of logical formalisms, rooted in classical first-order logic, widely used in Knowledge Representation and Reasoning to articulate and infer relationships among pertinent concepts within a specified application domain. It is widely utilized across various fields such as the semantic web [20, 3], ontologies [21], and software engineering [4]. Description logic offers solutions to diverse reasoning tasks arising from a knowledge base. Among the notable reasoning services offered by description logic is ontology-mediated query answering, which involves answering queries based on a given knowledge base [1, chpater 7].

In [7][1], a two-sorted *lattice-based description logic* LE-$\mathscr{ALC}$[2] was introduced based on non-distributive modal logic, with semantics grounded in an enriched formal context [12, 11]. LE-$\mathscr{ALC}$ provides a natural description logic to reason about formal concepts (or categories) arising from formal contexts in Formal Concept Analysis (FCA) [15, 16]. The logic LE-$\mathscr{ALC}$ has the same relationship with non-distributive modal logic and its semantics based on formal contexts as the relationship between $\mathscr{ALC}$ and the classical normal modal logic with its Kripke frame semantics. Namely, LE-$\mathscr{ALC}$ facilitates the description of *enriched formal contexts*, i.e., formal contexts endowed with additional relations, which give rise to concept lattices extended with normal modal operators. Similarly to the classical modal operators, the 'non-distributive' modal operators can be given different interpretations, such as the epistemic operator [11] and the approximation operator [10].

In this paper, we adapt and modify the LE-$\mathscr{ALC}$ tableaux algorithm provided in [7] to answer several different types of queries based on LE-$\mathscr{ALC}$ knowledge bases with acyclic TBoxes. We show

*Krishna Manoorkar is supported by the NWO grant KIVI.2019.001 awarded to Alessandra Palmigiano.

†Ruoding Wang is supported by the China Scholarship Council No.202206310072.

[1]We noticed a mistake in the proof of termination and I-compatibility in an earlier version of this paper [6] in which concepts $\top$ and $\bot$ were included as concept names. In the updated version [7] we prove that the result holds in the restriction which does not contain $\top$ and $\bot$ in the language of concept names. In this paper, we work with the restricted language as in [7].

[2]Even though concept names in LE-$\mathscr{ALC}$ do not contain negation, we still refer to this description logic as LE-$\mathscr{ALC}$ rather than LE-$\mathscr{ALE}$, as negation on ABox terms is included in the description logic language.

Cynthia Kop and Helida Salles Santos (Eds); 19th Workshop on
Logical and Semantic Frameworks with Applications (LSFA'24)
EPTCS 421, 2025, pp. 21–43, doi:10.4204/EPTCS.421.2

that for any consistent LE-$\mathscr{ALC}$ ABox $\mathscr{A}$, the model constructed from the tableaux completion of $\mathscr{A}$ is a universal or canonical model for answering different queries like *relationship queries* asking if an object and a feature are related, *membership queries* asking if an object or a feature belongs to a concept, and *subsumption queries* asking if a concept is included in some other concept. This allows us to answer multiple such queries in polynomial time in $|\mathscr{A}|$. We show that it also acts as a universal model w.r.t. negative relational queries, however this is not true for negative membership or subsumption queries.

Finally, we consider separation queries which ask if two objects or features can be distinguished from each other by means of some role (relation). We convert these queries into an equivalent problem of checking the consistency of the given ABox w.r.t. some extension of LE-$\mathscr{ALC}$ and providing a tableaux algorithm for such extension. This method allows to answer separation queries of different types in polynomial time in $|\mathscr{A}|$.

Structure of the paper. In Section 2, we briefly review non-distributive modal logic and polarity-based semantics, lattice-based description logic LE-$\mathscr{ALC}$, and the tableaux algorithm for checking its ABox consistency. In Section 3, we demonstrate that the model obtained from the Tableaux Algorithm (Section 2) is a universal model for various queries, and we define different types of queries and corresponding algorithms. Section 4 provides a specific LE-$\mathscr{ALC}$ knowledge base and illustrates how the algorithms answer the queries discussed earlier. Finally, Section 5 summarizes the paper and outlines future directions.

2 Preliminaries

In this section, we collect preliminaries on non-distributive modal logic and its polarity-based semantics, i.e. semantics based on formal contexts, and the lattice-based description logic LE-$\mathscr{ALC}$ with the tableaux algorithm developed for it in [7].

2.1 Basic non-distributive modal logic and its polarity-based semantics

In this section, we briefly introduce the basic non-distributive modal logic and polarity-based semantics for it. It is a member of a family of lattice-based logics, sometimes referred to as *LE-logics* (cf. [13]), which have been studied in the context of a research program on the logical foundations of categorization theory [12, 11, 10, 14]. Let Prop be a (countable) set of atomic propositions. The language $\mathscr{L}$ is defined as follows:

$$\varphi := \bot \mid \top \mid p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi,$$

where $p \in \mathsf{Prop}$. The *basic*, or *minimal normal $\mathscr{L}$-logic* is a set $\mathbf{L}$ of sequents $\varphi \vdash \psi$, with $\varphi, \psi \in \mathscr{L}$, containing the following axioms:

$$\begin{array}{lllll} p \vdash p & \bot \vdash p & p \vdash p \vee q & p \wedge q \vdash p & \top \vdash \Box\top \quad\quad \Box p \wedge \Box q \vdash \Box(p \wedge q) \\ & p \vdash \top & q \vdash p \vee q & p \wedge q \vdash q & \Diamond\bot \vdash \bot \quad\quad \Diamond(p \vee q) \vdash \Diamond p \vee \Diamond q \end{array}$$

and closed under the following inference rules:

$$\frac{\varphi \vdash \chi \quad \chi \vdash \psi}{\varphi \vdash \psi} \quad \frac{\varphi \vdash \psi}{\varphi(\chi/p) \vdash \psi(\chi/p)} \quad \frac{\chi \vdash \varphi \quad \chi \vdash \psi}{\chi \vdash \varphi \wedge \psi} \quad \frac{\varphi \vdash \chi \quad \psi \vdash \chi}{\varphi \vee \psi \vdash \chi} \quad \frac{\varphi \vdash \psi}{\Box\varphi \vdash \Box\psi} \quad \frac{\varphi \vdash \psi}{\Diamond\varphi \vdash \Diamond\psi}$$

In the following part, we define the polarity-based semantics for this logic.

Relational semantics. The following preliminaries are taken from [10, 14]. For any binary relation $T \subseteq U \times V$, and any $U' \subseteq U$ and $V' \subseteq V$, we let

$$T^{(1)}[U'] := \{v \mid \forall u(u \in U' \Rightarrow uTv)\} \qquad T^{(0)}[V'] := \{u \mid \forall v(v \in V' \Rightarrow uTv)\}.$$

For any $u \in U$ (resp. $v \in V$) we will write $T^{(1)}[u]$ (resp. $T^{(0)}[v]$) in place of $T^{(1)}[\{u\}]$ (resp. $T^{(0)}[\{v\}]$).

A *polarity* or *formal context* (cf. [16]) is a tuple $\mathbb{P} = (A, X, I)$, where A and X are sets, and $I \subseteq A \times X$ is a binary relation. A and X can be understood as the collections of *objects* and *features*, and for any $a \in A$ and $x \in X$, aIx exactly when the object a has the feature x. For any polarity $\mathbb{P} = (A, X, I)$, the pair of maps

$$(\cdot)^{\uparrow} : \mathscr{P}(A) \to \mathscr{P}(X) \text{ and } (\cdot)^{\downarrow} : \mathscr{P}(X) \to \mathscr{P}(A),$$

defined by $B^{\uparrow} := I^{(1)}[B]$ and $Y^{\downarrow} := I^{(0)}[Y]$ where $B \subseteq A$ and $Y \subseteq X$, forms a Galois connection, and hence induces the closure operators $(\cdot)^{\uparrow\downarrow}$ and $(\cdot)^{\downarrow\uparrow}$ on $\mathscr{P}(A)$ and on $\mathscr{P}(X)$, respectively. Again, we will write $a^{\uparrow}$ and $a^{\uparrow\downarrow}$ (resp. $x^{\downarrow}$ and $x^{\downarrow\uparrow}$) in place of $\{a\}^{\uparrow}$ and $\{a\}^{\uparrow\downarrow}$ (resp. $\{x\}^{\downarrow}$ and $\{x\}^{\downarrow\uparrow}$).

A *formal concept* of a polarity $\mathbb{P} = (A, X, I)$ is a tuple $c = (\llbracket c \rrbracket, (\!|c|\!))$ such that $\llbracket c \rrbracket \subseteq A$ and $(\!|c|\!) \subseteq X$, and $\llbracket c \rrbracket = (\!|c|\!)^{\downarrow}$ and $(\!|c|\!) = \llbracket c \rrbracket^{\uparrow}$, i.e. the sets $\llbracket c \rrbracket$ and $(\!|c|\!)$ are Galois-stable. The set of formal concepts of polarity $\mathbb{P}$, with the order defined by

$$c_1 \leq c_2 \quad \text{iff} \quad \llbracket c_1 \rrbracket \subseteq \llbracket c_2 \rrbracket \quad \text{iff} \quad (\!|c_2|\!) \subseteq (\!|c_1|\!),$$

forms a complete lattice $\mathbb{P}^+$, namely the *concept lattice* of $\mathbb{P}$.

An *enriched formal context* is a tuple $\mathbb{F} = (\mathbb{P}, R_{\Box}, R_{\Diamond})$, where $R_{\Box} \subseteq A \times X$ and $R_{\Diamond} \subseteq X \times A$ are *I-compatible* relations, that is, for all $a \in A$ and $x \in X$, the sets $R_{\Box}^{(0)}[x]$, $R_{\Box}^{(1)}[a]$, $R_{\Diamond}^{(0)}[a]$, $R_{\Diamond}^{(1)}[x]$ are Galois-stable in $\mathbb{P}$. Given the operations $[R_{\Box}]$ and $\langle R_{\Diamond}\rangle$ on $\mathbb{P}^+$ corresponding to $R_{\Box}$ and $R_{\Diamond}$, respectively, we have for any $c \in \mathbb{P}^+$,

$$[R_{\Box}]c = (R_{\Box}^{(0)}[(\!|c|\!)], I^{(1)}[R_{\Box}^{(0)}[(\!|c|\!)]]) \quad \text{and} \quad \langle R_{\Diamond}\rangle c = (I^{(0)}[R_{\Diamond}^{(0)}[\llbracket c \rrbracket]], R_{\Diamond}^{(0)}[\llbracket c \rrbracket]).$$

We refer to the algebra $\mathbb{F}^+ = (\mathbb{P}^+, [R_{\Box}], \langle R_{\Diamond}\rangle)$ as the *complex algebra* of $\mathbb{F}$. A *valuation* on such an $\mathbb{F}$ is a map $V\colon \mathsf{Prop} \to \mathbb{P}^+$. For each $p \in \mathsf{Prop}$, we let $\llbracket p \rrbracket := \llbracket V(p) \rrbracket$ (resp. $(\!|p|\!) := (\!|V(p)|\!)$) denote the extension (resp. intension) of the interpretation of p under V.

A *model* is a tuple $\mathbb{M} = (\mathbb{F}, V)$, where $\mathbb{F} = (\mathbb{P}, R_{\Box}, R_{\Diamond})$ is an enriched formal context and V is a valuation of $\mathbb{F}$. For every $\varphi \in \mathscr{L}$, we let $\llbracket \varphi \rrbracket_{\mathbb{M}} := \llbracket V(\varphi) \rrbracket$ (resp. $(\!|\varphi|\!)_{\mathbb{M}} := (\!|V(\varphi)|\!)$) denote the extension (resp. intension) of the interpretation of φ under the homomorphic extension of V. The 'satisfaction' and 'co-satisfaction' relations $\Vdash$ and $\succ$ can be recursively defined as follows:

$\mathbb{M}, a \Vdash p$	iff $a \in \llbracket p \rrbracket_{\mathbb{M}}$	$\mathbb{M}, x \succ p$	iff $x \in (\!|p|\!)_{\mathbb{M}}$
$\mathbb{M}, a \Vdash \top$	always	$\mathbb{M}, x \succ \top$	iff aIx for all $a \in A$
$\mathbb{M}, x \succ \bot$	always	$\mathbb{M}, a \Vdash \bot$	iff aIx for all $x \in X$
$\mathbb{M}, a \Vdash \varphi \wedge \psi$	iff $\mathbb{M}, a \Vdash \varphi$ and $\mathbb{M}, a \Vdash \psi$	$\mathbb{M}, x \succ \varphi \wedge \psi$	iff $(\forall a \in A)$ $(\mathbb{M}, a \Vdash \varphi \wedge \psi \Rightarrow aIx)$
$\mathbb{M}, x \succ \varphi \vee \psi$	iff $\mathbb{M}, x \succ \varphi$ and $\mathbb{M}, x \succ \psi$	$\mathbb{M}, a \Vdash \varphi \vee \psi$	iff $(\forall x \in X)$ $(\mathbb{M}, x \succ \varphi \vee \psi \Rightarrow aIx)$.

As to the interpretation of modal operators:

$\mathbb{M}, a \Vdash \Box\varphi$	iff $(\forall x \in X)(\mathbb{M}, x \succ \varphi \Rightarrow aR_{\Box}x)$	$\mathbb{M}, x \succ \Box\varphi$	iff $(\forall a \in A)(\mathbb{M}, a \Vdash \Box\varphi \Rightarrow aIx)$
$\mathbb{M}, x \succ \Diamond\varphi$	iff $(\forall a \in A)(\mathbb{M}, a \Vdash \varphi \Rightarrow xR_{\Diamond}a)$	$\mathbb{M}, a \Vdash \Diamond\varphi$	iff $(\forall x \in X)(\mathbb{M}, x \succ \Diamond\varphi \Rightarrow aIx)$.

The definition above ensures that, for any $\mathscr{L}$-formula φ,

$$\mathbb{M}, a \Vdash \varphi \quad \text{iff} \quad a \in \llbracket \varphi \rrbracket_{\mathbb{M}}, \qquad \text{and} \qquad \mathbb{M}, x \succ \varphi \quad \text{iff} \quad x \in (\!|\varphi|\!)_{\mathbb{M}}.$$
$$\mathbb{M} \models \varphi \vdash \psi \quad \text{iff} \quad \llbracket \varphi \rrbracket_{\mathbb{M}} \subseteq \llbracket \psi \rrbracket_{\mathbb{M}} \quad \text{iff} \quad (\!|\psi|\!)_{\mathbb{M}} \subseteq (\!|\varphi|\!)_{\mathbb{M}}.$$

The interpretation of the propositional connectives $\vee$ and $\wedge$ in the framework described above reproduces the standard notion of join and meet of formal concepts used in FCA. The interpretation of operators $\Box$ and $\Diamond$ is motivated by algebraic properties and duality theory for modal operators on lattices (see [14, Section 3] for an expanded discussion). In [10, Proposition 3.7], it is shown that the semantics of LE-logics is compatible with Kripke semantics for classical modal logic, and thus, LE-logics are indeed generalizations of classical modal logic. This interpretation is further justified in [10, Section 4] by noticing that, under the interpretations of the relation I as aIx iff "object a has feature x" and

$R = R_\Box = R_\Diamond^{-1}$ as aRx iff "there is evidence that object a has feature x", then, for any concept c, the extents of concepts $\Box c$ and $\Diamond c$ can be interpreted as "the set of objects which *certainly* belong to c" (upper approximation), and "the set of objects which *possibly* belong to c" (lower approximation) respectively. Thus, the interpretations of $\Box$ and $\Diamond$ have similar meaning in the LE-logic as in the classical modal logic.

2.2 Description logic LE-$\mathscr{ALC}$

In this section, we recall the lattice-based description logic LE-$\mathscr{ALC}$ introduced in [7] as a counterpart of non-distributive modal logic. It serves as a natural framework in the realm of description logics for reasoning about the (enriched) formal contexts and the concepts defined by them.

The language of LE-$\mathscr{ALC}$ contains two types of individuals, usually interpreted as *objects* and *features*. Let OBJ and FEAT be disjoint sets of individual names for objects and features. The set $\mathscr{R}$ of the role names for LE-$\mathscr{ALC}$ is the union of three types of relations: (1) a unique relation $I \subseteq \mathsf{OBJ} \times \mathsf{FEAT}$; (2) a set of relations $\mathscr{R}_\Box$ of the form $R_\Box \subseteq \mathsf{OBJ} \times \mathsf{FEAT}$; (3) a set of relations $\mathscr{R}_\Diamond$ of the form $R_\Diamond \subseteq \mathsf{FEAT} \times \mathsf{OBJ}$. The relation I is intended to be interpreted as the incidence relation of formal contexts and encodes information on which objects have which features, and the relations in $\mathscr{R}_\Box$ and $\mathscr{R}_\Diamond$ encode additional relationships between objects and features (see [10] for an extended discussion). In this paper, we work with an LE-$\mathscr{ALC}$ language in which the sets of role names $\mathscr{R}_\Box$ and $\mathscr{R}_\Diamond$ are singletons. All the results in this paper can be generalized to language with multiple role names in each of these sets straightforwardly.

For any set $\mathscr{D}$ of atomic concept names, the language of LE-$\mathscr{ALC}$ concepts is:

$$C := D \mid C_1 \wedge C_2 \mid C_1 \vee C_2 \mid \langle R_\Diamond \rangle C \mid [R_\Box] C$$

where $D \in \mathscr{D}$. This language matches the LE-logic language and has an analogous intended interpretation of the complex algebras of the enriched formal contexts (cf. Section 2.1). As usual in FCA, $\vee$ and $\wedge$ are to be interpreted as the smallest common superconcept and the greatest common subconcept. We do not use the symbols $\forall r$ and $\exists r$ in the context of LE-$\mathscr{ALC}$ because using the same notation verbatim would be ambiguous or misleading, as the semantic clauses of modal operators in LE-logic use the universal quantifiers.

TBox assertions in LE-$\mathscr{ALC}$ are of the shape $C_1 \equiv C_2$, where C_1 and C_2 are concepts defined as above. As is standard in DL (see [1] for more details), general concept inclusions of the form $C_1 \sqsubseteq C_2$ can be rewritten as $C_1 \equiv C_2 \wedge C_3$, where C_3 is a new concept name. ABox assertions are of the form:

$$aR_\Box x, \quad xR_\Diamond a, \quad aIx, \quad a : C, \quad x :: C, \quad \neg\alpha,$$

where α is any of the first five ABox terms. We refer to the first three types of terms as *relational terms*. We denote an arbitrary ABox (resp. TBox) with $\mathscr{A}$ (resp. $\mathscr{T}$). The interpretations of the terms $a : C$ and $x :: C$ are: "object a is a member of concept C", and "feature x is in the description of concept C", respectively. Note that we explicitly add negative terms to ABoxes, as the concept names in LE-$\mathscr{ALC}$ do not contain negations.

An *interpretation* for LE-$\mathscr{ALC}$ is a tuple $\mathscr{M} = (\mathbb{F}, \cdot^{\mathscr{M}})$, where $\mathbb{F} = (\mathbb{P}, R_\Box, R_\Diamond)$ is an enriched formal context, and $\cdot^{\mathscr{M}}$ maps:

1. individual names $a \in \mathsf{OBJ}$ (resp. $x \in \mathsf{FEAT}$) to some $a^{\mathscr{M}} \in A$ (resp. $x^{\mathscr{M}} \in X$);
2. role names I, $R_\Box$ and $R_\Diamond$ to relations $I^{\mathscr{M}} \subseteq A \times X$, $R_\Box^{\mathscr{M}} \subseteq A \times X$ and $R_\Diamond^{\mathscr{M}} \subseteq X \times A$ in $\mathbb{F}$;
3. any atomic concept D to $D^{\mathscr{M}} \in \mathbb{F}^+$, and other concepts as follows:

$(C_1 \wedge C_2)^{\mathscr{M}} = C_1^{\mathscr{M}} \wedge C_2^{\mathscr{M}} \quad (C_1 \vee C_2)^{\mathscr{M}} = C_1^{\mathscr{M}} \vee C_2^{\mathscr{M}} \quad ([R_\Box]C)^{\mathscr{M}} = [R_\Box^{\mathscr{M}}]C^{\mathscr{M}} \quad (\langle R_\Diamond \rangle C)^{\mathscr{M}} = \langle R_\Diamond^{\mathscr{M}} \rangle C^{\mathscr{M}}$

where all the connectives are interpreted as defined in LE-logic (cf. Section 2.1). The satisfiability relation for an interpretation $\mathscr{M}$ is defined as follows:

1. $\mathscr{M} \models C_1 \equiv C_2$ iff $[\![C_1^{\mathscr{M}}]\!] = [\![C_2^{\mathscr{M}}]\!]$ iff $(\![C_2^{\mathscr{M}}]\!) = (\![C_1^{\mathscr{M}}]\!)$.

2. $\mathscr{M} \models a : C$ iff $a^{\mathscr{M}} \in [\![C^{\mathscr{M}}]\!]$ and $\mathscr{M} \models x :: C$ iff $x^{\mathscr{M}} \in ([C^{\mathscr{M}}])$.
3. $\mathscr{M} \models aIx$ (resp. $aR_{\Box}x$, $xR_{\Diamond}a$) iff $a^{\mathscr{M}} I^{\mathscr{M}} x^{\mathscr{M}}$ (resp. $a^{\mathscr{M}} R_{\Box}^{\mathscr{M}} x^{\mathscr{M}}$, $x^{\mathscr{M}} R_{\Diamond}^{\mathscr{M}} a^{\mathscr{M}}$).
4. $\mathscr{M} \models \neg\alpha$, where α is any ABox term, iff $\mathscr{M} \not\models \alpha$.

The satisfaction definition can be extended to concept inclusion as follows. For any concepts C_1, and C_2, and an interpretation $\mathscr{M}$, $\mathscr{M} \models C_1 \sqsubseteq C_2$ iff $C_1^{\mathscr{M}} \leq C_2^{\mathscr{M}}$.

An interpretation $\mathscr{M}$ is a *model* for an LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ knowledge base $(\mathscr{A}, \mathscr{T})$, where $\mathscr{A}$ is an ABox and $\mathscr{T}$ is a TBox, if $\mathscr{M} \models \mathscr{A}$ and $\mathscr{M} \models \mathscr{T}$. An LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ knowledge base $(\mathscr{A}, \mathscr{T})$ is said to be inconsistent if there is no model for it. We say an ABox $\mathscr{A}$ is consistent if knowledge base $(\mathscr{A}, \mathscr{T})$ has a model and $\mathscr{T}$ is empty. In this paper, we use LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ knowledge bases to mean LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ knowledge bases with acyclic Tboxes unless otherwise stated.

2.3 Tableaux algorithm for checking LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox consistency

In this section, we introduce the tableaux algorithm for checking the consistency of LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABoxes. An LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$ contains a *clash* iff it contains both β and $\neg\beta$ for some relational term β. The expansion rules below are designed so that the expansion of $\mathscr{A}$ will contain a clash iff $\mathscr{A}$ is inconsistent. The set $sub(C)$ of sub-formulas of any LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ concept name C is defined as usual. A concept name C' *occurs* in the ABox $\mathscr{A}$ (denoted as $C' \in \mathscr{A}$) if $C' \in sub(C)$ for some C such that one of the terms $a : C$, $x :: C$, $\neg(a : C)$, or $\neg(x :: C)$ is in $\mathscr{A}$. A constant b (resp. y) *occurs* in $\mathscr{A}$ ($b \in \mathscr{A}$, or $y \in \mathscr{A}$), iff some term containing b (resp. y) occurs in it.

The tableaux algorithm below provides a method to construct a model $(\mathbb{F}, \cdot^{\mathscr{M}})$ for every consistent $\mathscr{A}$, where $\mathbb{F} = (\mathbb{P}, R_{\Box}, R_{\Diamond})$ is such that, for any $C \in \mathscr{A}$, some $a_C \in A$ and $x_C \in X$ exist such that, for any $a \in A$ (resp. any $x \in X$), $a \in [\![C^{\mathscr{M}}]\!]$ (resp. $x \in ([C^{\mathscr{M}}])$) iff aIx_C (resp. $a_C Ix$). We call a_C and x_C the *classifying object* and the *classifying feature* of C, respectively. To make the notation easily readable, we write $a_{\Box C}$, $x_{\Box C}$ (resp. $a_{\Diamond C}$, $x_{\Diamond C}$) instead of $a_{[R_{\Box}]C}$, $x_{[R_{\Box}]C}$ (resp. $a_{\langle R_{\Diamond}\rangle C}$, $x_{\langle R_{\Diamond}\rangle C}$). The commas in each rule are meta-linguistic conjunctions, hence every tableau is non-branching.

Algorithm 1 tableaux algorithm for checking LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox consistency

Input: An LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$. **Output**: whether $\mathscr{A}$ is inconsistent.

1: **if** there is a clash in $\mathscr{A}$ **then return** "inconsistent".
2: **pick** any applicable expansion rule R, **apply** R to $\mathscr{A}$ and proceed recursively.
3: **if** there is no clash after post-processing **return** "consistent".

Creation rule

$$\text{create}\ \frac{\text{For any } C \in \mathscr{A}}{a_C : C, \quad x_C :: C}$$

Basic rule

$$I\ \frac{b : C, \quad y :: C}{bIy}$$

Rules for the logical connectives

$$\frac{b : C_1 \wedge C_2}{b : C_1, \quad b : C_2}\ \wedge_A \qquad \vee_X\ \frac{y :: C_1 \vee C_2}{y :: C_1, \quad y :: C_2}$$

$$\Box\ \frac{b : [R_{\Box}]C, \quad y :: C}{bR_{\Box}y} \qquad \frac{y :: \langle R_{\Diamond}\rangle C, \quad b : C}{yR_{\Diamond}b}\ \Diamond$$

***I*-compatibility rules**

$$\Box_y\ \frac{bI\Box y}{bR_{\Box}y} \qquad \frac{bI\blacksquare y}{yR_{\Diamond}b}\ \blacksquare_y$$

$$\Diamond_b\ \frac{\Diamond bIy}{yR_{\Diamond}b} \qquad \frac{\blacklozenge bIy}{bR_{\Box}y}\ \blacklozenge_b$$

inverse rules for connectives

$$\wedge_A^{-1}\ \frac{b : C_1, b : C_2, C_1 \wedge C_2 \in \mathscr{A}}{b : C_1 \wedge C_2} \qquad \frac{y :: C_1, y :: C_2, C_1 \vee C_2 \in \mathscr{A}}{y :: C_1 \vee C_2}\ \vee_X^{-1}$$

Adjunction rules

$$R_\Box\ \frac{bR_\Box y}{\blacklozenge bIy,\quad bI\Box y} \qquad \frac{yR_\Diamond b}{\Diamond bIy,\quad bI\blacksquare y}\ R_\Diamond$$

Basic rules for negative assertions **Appending rules**

$$\neg b\ \frac{\neg(b:C)}{\neg(bIx_C)} \qquad \frac{\neg(x::C)}{\neg(a_CIx)}\ \neg x \qquad x_C\ \frac{bIx_C}{b:C} \qquad \frac{a_CIy}{y::C}\ a_C$$

Note that in the creation rule, the a_C and x_C are new (i.e. different from any names already appearing in the tableaux) special object and feature names unique for each C. In the adjunction rules, the individuals $\blacklozenge b$, $\Box y$, $\Diamond b$, and $\blacksquare y$ are new and unique individual names[3] for relations $R_\Box$ and $R_\Diamond$, and individuals b and y, except for $\Diamond a_C = a_{\Diamond C}$ and $\Box x_C = x_{\Box C}$. Side conditions that the conjunction and disjunction occur in $\mathscr{A}$ for rules $\wedge_A^{-1}$ and $\vee_X^{-1}$ ensure that we do not add new meets or joins to the concept names.

The following theorem follows from the results in [7]:

For any consistent LE-$\mathscr{ALC}$ ABox $\mathscr{A}$, the tableaux completion $\overline{\mathscr{A}}$ of $\mathscr{A}$ is a set of assertions which are obtained by applying the tableaux algorithm 1 to $\mathscr{A}$. From $\overline{\mathscr{A}}$, we can construct a model $\mathscr{M} = (\mathbb{F}, \cdot^{\mathscr{M}})$, where $\mathbb{F} = (A, X, I, R_\Box, R_\Diamond)$ is described as follows: A and X are taken to be the sets of all individual names of object and feature that occur in $\overline{\mathscr{A}}$, respectively, and all individuals are interpreted by their names. For any role name R, its interpretation $R^{\mathscr{M}}$ is defined as follows: for any individual names l, m, $lR^{\mathscr{M}}m$ iff $lRm \in \overline{\mathscr{A}}$. Finally, for the atomic concept D, its interpretation is set to the concept $(x_D^{\downarrow}, a_D^{\uparrow})$. The following result was proved in [7].

Theorem 1. *For any LE-$\mathscr{ALC}$ ABox $\mathscr{A}$,*

- *the tableaux algorithm applied to $\mathscr{A}$ terminates in polynomial time in $|\mathscr{A}|$;*
- *$\overline{\mathscr{A}}$ contains a clash iff $\mathscr{A}$ is inconsistent;*
- *if $\mathscr{A}$ is inconsistent, then the model $\mathscr{M}$ as constructed above is a model for $\mathscr{A}$ of the size polynomial in $|\mathscr{A}|$. Moreover, for any individual names b, y, and concept C occurring in $\mathscr{A}$, $b \in [\![C]\!]$ iff $bIx_C \in \overline{\mathscr{A}}$, and $y \in ([C])$ iff $a_cIy \in \overline{\mathscr{A}}$.*

Remark 1. *The algorithm can be easily extended to acyclic TBoxes (exponential-time), using the unraveling technique (see [2] for details).*

2.4 Ontology-mediated query answering

A key task in description logic ontologies (knowledge bases) is to support various reasoning tasks, one of which is to answer queries based on ontologies [1, chapter 7]. Let $\mathscr{K} = (\mathscr{A}, \mathscr{T})$ be a consistent knowledge base given in a specific description logic DL. Given a query $q(\overline{p})$ (with a possibly empty tuple of free variables $\overline{p}$) in (appropriate) first-order language and a model $\mathscr{M}$ of $\mathscr{K}$, we say that a sequence of individuals $\overline{a}$ in $\mathscr{A}$ is an *answer for query* $q(\overline{p})$ w.r.t. model $\mathscr{M}$ of knowledge base $\mathscr{K}$ if $\mathscr{M} \models q(\overline{a})$. An answer $\overline{a}$ in $\mathscr{A}$ is said to be *a certain answer for the query* $q(\overline{p})$ with respect to a knowledge base $\mathscr{K}$ if it is an answer for $q(\overline{p})$ w.r.t. all the models of $\mathscr{K}$. An important notion used in ontology-mediated query answering is that of a *universal* or *canonical* model. For a query $q(\overline{p})$ on a knowledge base $\mathscr{K}$, we say that a model $\mathscr{M}$ of $\mathscr{K}$ is a universal or canonical model for $\mathscr{K}$ if for any $\overline{a}$ appearing in $\mathscr{K}$, $\mathscr{K} \models q(\overline{a})$ iff $\mathscr{M} \models q(\overline{a})$. In case $\overline{p}$ is empty, the answer or the certain answer for

[3] The new individual names $\blacklozenge b$, $\Diamond b$, $\Box y$, and $\blacksquare y$ appearing in tableaux expansion are purely syntactic entities. Intuitively, they correspond to the classifying objects (resp. features) of the concepts $\blacklozenge\mathbf{b}$, $\Diamond\mathbf{b}$ (resp. $\Box\mathbf{y}$, resp. $\blacksquare\mathbf{y}$), where $\mathbf{b} = (b^{\uparrow\downarrow}, b^{\uparrow})$ (resp. $\mathbf{y} = (y^{\downarrow}, y^{\downarrow\uparrow})$) is the concept generated by b (resp. y), and the operation $\blacklozenge$ (resp.$\blacksquare$) is the left (resp. right) adjoint of operation $\Box$ (resp. $\Diamond$).

such query is true or false depending on whether $\mathscr{M} \models q$ or not. Thus, we can provide certain answer for query q over $\mathscr{K}$ by only looking over the universal or canonical model $\mathscr{M}$. Universal models for different description logics have been extensively studied [9, 17, 19, 8]. In this paper, we would focus on answering some specific types of queries over knowledge bases in non-distributive description logic LE-$\mathscr{A}\mathscr{L}\mathscr{C}$. To this end, we show that for any LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$, the model constructed from it by applying the LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ tableaux algorithm 1 acts as the universal model for $\mathscr{A}$ w.r.t. several different types of queries. As the tableaux algorithm is polynomial in time and produces a polynomial size model in $|\overline{\mathscr{A}}|$, this provides a polynomial-time algorithm to answer these types of queries.

3 Query answering over LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABoxes

In this section, we discuss different types of queries pertaining to LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABoxes and develop algorithms to answer them. We start by showing that for any consistent LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$, the model obtained using Algorithm 1 behaves like universal model w.r.t. several types of queries.

3.1 Universal model for LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox

For any individual name appearing in the tableaux expansion $\overline{\mathscr{A}}$ of an LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$, we define its *concept companion* as follows :

1. For any constant b (resp. y) appearing in $\mathscr{A}$, $con(b)$ (resp. $con(y)$) is a concept such that for any interpretation $\mathscr{M}$, $con(b)^{\mathscr{M}} = \mathbf{b}$ (resp. $con(y)^{\mathscr{M}} = \mathbf{y}$), where $\mathbf{b}$ (resp. $\mathbf{y}$) denotes the concept generated by $b^{\mathscr{M}}$ (resp. $y^{\mathscr{M}}$), i.e. $\mathbf{b} = ((b^{\mathscr{M}})^{\uparrow\downarrow}, (b^{\mathscr{M}})^{\uparrow})$ (resp. $\mathbf{y} = ((y^{\mathscr{M}})^{\downarrow}, (y^{\mathscr{M}})^{\downarrow\uparrow})$).

2. For any constant $\Diamond b$ (resp. $\blacklozenge b$, resp. $\blacksquare y$, resp. $\Box y$) appearing in $\mathscr{A}$, $con(\Diamond b) = \Diamond con(b)$ (resp. $con(\blacklozenge b) = \blacklozenge con(b)$, resp. $con(\blacksquare y) = \blacksquare con(y)$, resp. $con(\Box y) = \Box con(y)$), where the operation $\blacklozenge$ (resp. $\blacksquare$) is the left (resp. right) adjoint of $\Box$ (resp. $\Diamond$).

Lemma 1. *For any consistent LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$, individual names b, y appearing in its completion $\overline{\mathscr{A}}$, and concept C appearing in $\mathscr{A}$:*

1. $\mathscr{A} \models con(b) \sqsubseteq con(y)$ *iff* $bIy \in \overline{\mathscr{A}}$, *2.* $\mathscr{A} \models con(b) \sqsubseteq \Box con(y)$ *iff* $bR_{\Box}y \in \overline{\mathscr{A}}$,
3. $\mathscr{A} \models \Diamond con(b) \sqsubseteq con(y)$ *iff* $yR_{\Diamond}b \in \overline{\mathscr{A}}$, *4.* $\mathscr{A} \models con(b) \sqsubseteq C$ *iff* $bIx_C \in \overline{\mathscr{A}}$,
5. $\mathscr{A} \models C \sqsubseteq con(y)$ *iff* $a_C Iy \in \overline{\mathscr{A}}$, *6.* $\mathscr{A} \models con(b) \sqsubseteq C$ *iff* $b : C \in \overline{\mathscr{A}}$,
7. $\mathscr{A} \models C \sqsubseteq con(y)$ *iff* $y :: C \in \overline{\mathscr{A}}$, *8.* $\mathscr{A} \models C_1 \sqsubseteq C_2$ *iff* $a_{C_1} Ix_{C_2} \in \overline{\mathscr{A}}$.

Proof. The proofs from left to right for items 1-5 follow immediately from Theorem 1. We prove the right to left implications by simultaneous (over all the items) induction on the number of expansion rules applied. The base case is when the term in the right appears in $\mathscr{A}$. In this case, it is immediate from the definition that we get the required condition on the left.

Creation rule. By this rule, $a_C : C$ and $x_C :: C$ are added by any $C \in \mathscr{A}$, which imply $C \sqsubseteq C$.

Basic rule. By this rule, bIy is added from $b : C$ and $y :: C$. By induction applied to items 6 and 7, we get $con(b) \sqsubseteq C$ and $C \sqsubseteq con(y)$, which imply that $con(b) \sqsubseteq con(y)$. It is easy to check item 4 and item 5 also hold. For item 8, $a_{C_1} Ix_{C_2}$ is added from $a_{C_1} : C$ and $x_{C_2} :: C$. By induction applied to items 6 and 7, we have $C_1 \sqsubseteq C$ and $C \sqsubseteq C_2$, which imply that $C_1 \sqsubseteq C_2$.

Rules $\wedge_A$, $\vee_X$, $\wedge_A^{-1}$, $\vee_X^{-1}$. We give the proofs for rules $\wedge_A$ and $\wedge_A^{-1}$. The proofs for $\vee_X$ and $\vee_X^{-1}$ are analogous. By rule $\wedge_A$, $b : C_1$ and $b : C_2$ are added from $b : C_1 \wedge C_2$. By induction applied to item 6, $con(b) \sqsubseteq C_1 \wedge C_2$, and thus $con(b) \sqsubseteq C_1$ and $con(b) \sqsubseteq C_2$. By rule $\wedge_A^{-1}$, $b : C_1 \wedge C_2$ is added from $b : C_1$, $b : C_2$, and $C_1 \wedge C_2 \in \mathscr{A}$. By induction applied to item 6, we have $con(b) \sqsubseteq C_1$ and $con(b) \sqsubseteq C_2$. Since $C_1 \wedge C_2$ exists in $\mathscr{A}$, we get $con(b) \sqsubseteq C_1 \wedge C_2$.

I-compatibility rules. We give the proofs for rules $\Box y$ and $\blacksquare y$. The proofs for $\Diamond b$ and $\blacklozenge b$ are analogous. By rule $\Box y$, $bR_{\Box}y$ is added from $bI\Box y$, by induction applied to item 1, we get $con(b) \sqsubseteq con(\Box y)$ and by definition we have $con(b) \sqsubseteq \Box con(y)$. By rule $\blacksquare y$, $yR_{\Diamond}b$ is added from $bI\blacksquare y$. By induction applied to item 1, we have $con(b) \sqsubseteq con(\blacksquare y)$, and by definition $con(b) \sqsubseteq \blacksquare con(y)$. By adjunction, we have $\Diamond con(b) \sqsubseteq con(y)$.

Rules $\Box$ and $\Diamond$. We give the proof for rule $\Box$, and the proof for $\Diamond$ is analogous. By rule $\Box$, $bR_{\Box}y$ is added from $b : [R_{\Box}]C$ and $y :: C$. By induction applied to item 6, we have $con(b) \sqsubseteq \Box C$. By induction on item 7, we get $C \sqsubseteq con(y)$. As $\Box$ is a monotone operator, we have $\Box C \sqsubseteq \Box con(y)$. Thus, $con(b) \sqsubseteq \Box con(y)$.

Adjunction rules. We give the proof for rule $R_{\Box}$ and the proof for $R_{\Diamond}$ is analogous. By rule $R_{\Box}$, $bI\Box y$ (resp. $\blacklozenge bIy$) is added from $bR_{\Box}y$. By induction applied to item 2, we have $con(b) \sqsubseteq \Box con(y)$ (resp. $\blacklozenge con(b) \sqsubseteq con(y)$ by adjunction), and thus $con(b) \sqsubseteq con(\Box y)$ (resp. $con(\blacklozenge b) \sqsubseteq con(y)$).

Appending rules. By rule x_C, the term $b : C$ is added from bIx_C. By induction applied to item 4, we have $con(b) \sqsubseteq C$. By rule a_C, $y :: C$ is added from $a_C Iy$ and by induction applied to item 5, we have $C \sqsubseteq con(y)$. □

Lemma 1 implies that for any consistent ABox $\mathscr{A}$, the model generated from $\mathscr{A}$ using Algorithm 1 acts as a universal model for several types of queries. We describe some such queries below.

Relationship queries. These queries are either Boolean queries asking if two individuals are related by relation I, $R_{\Box}$ or $R_{\Diamond}$, e.g. $q = bIy$, or queries asking for names of all individuals appearing in $\mathscr{A}$ that are related to some element by relation I, $R_{\Box}$ or $R_{\Diamond}$, e.g. $q(p) = bR_{\Box}p$.

Membership queries. These queries are either Boolean queries asking if some object or feature belongs to a given concept, e.g. $q = y :: C$, or queries asking for names of all individuals appearing in $\mathscr{A}$ that are in the extension or intension of a concept C, e.g. $q(p) = p : C$.

Subsumption queries. These queries are Boolean queries asking if a concept C_1 is included in C_2, i.e. $q = C_1 \sqsubseteq C_2$.[4]

As Algorithm 1 is polynomial-time and gives a model which is of polynomial-size in $|\mathscr{A}|$, we have the following corollary.

Corollary 1. *For any LE-$\mathscr{ALC}$ ABox $\mathscr{A}$, a query q of the above forms consisting of concepts and individual names appearing in $\mathscr{A}$, can be answered in polynomial time in $|\mathscr{A}|$ using Algorithm 1.*

Remark 2. *Relationship, membership, and subsumption queries can also be answered in polynomial time by converting them into a problem of consistency checking (see [1] for more details). However, it involves performing tableaux expansion for each query, while our result implies that we can answer multiple Boolean and naming queries with a single run of tableaux algorithm.*

If a subsumption or membership query has concept C not appearing in $\mathscr{A}$, we can answer such query by adding C to $\mathscr{A}$ through creation rule i.e. adding terms $a_C : C$, and $x_C :: C$ to $\mathscr{A}$. If we have multiple queries consisting of concepts not appearing in $\mathscr{A}$, we can add all of these concepts simultaneously and answer all queries with a single run of the tableaux algorithm.

Disjunctive relationship and membership queries. A disjunctive relationship (resp. membership) query is formed by taking the disjunction of a finite number of relationship (resp. membership) queries,

[4]Note that no non-trivial subsumptions are implied by knowledge bases with acyclic TBoxes. However, we include such queries as the algorithm can be used to answer queries regarding trivial (those implied by logic) subsumption efficiently. Moreover, we believe that the algorithm extend ideas used to answer these queries may be used in future generalizations to knowledge bases with cyclic TBoxes.

e.g. $q = bIy \vee bIz$, and $q(p) = p : C_1 \vee p : C_2$[5]. The following lemma implies that we can answer such queries in LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ by answering each disjunct separately.

Lemma 2. *Let t_1 and t_2 be any LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox terms not containing negation. Then, for any consistent LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$, $\mathscr{A} \models t_1 \vee t_2$ iff $\mathscr{A} \models t_1$ or $\mathscr{A} \models t_2$.*

Proof. $\mathscr{A} \models t_1 \vee t_2$ iff $\mathscr{A} \cup \{\neg t_1, \neg t_2\}$ is inconsistent. By tableaux algorithm for LE-$\mathscr{A}\mathscr{L}\mathscr{C}$, we have $\overline{\mathscr{A} \cup \{\neg t_1, \neg t_2\}} = \overline{\mathscr{A}} \cup B$, where the only terms in B are $\neg t_1$, $\neg t_2$, and the terms obtained by applying the negative rule $\neg b$ or $\neg x$ to these terms. Therefore, as $\mathscr{A} \cup \{\neg t_1, \neg t_2\}$ is inconsistent, $\overline{\mathscr{A}} \cup B$ must contain a clash. But as $\mathscr{A}$ is consistent, $\overline{\mathscr{A}}$ does not contain any clash. Therefore, some term in $\overline{\mathscr{A}}$ clashes with $\neg t_1$ or $\neg t_2$ or the term obtained by applying negative rule $\neg b$ or $\neg x$ to these terms. This implies that $\mathscr{A} \cup \{\neg t_1\}$ or $\mathscr{A} \cup \{\neg t_1\}$ must be inconsistent. Therefore, we have $\mathscr{A} \models t_1$ or $\mathscr{A} \models t_2$. □

3.2 Negative queries

Negative queries are obtained by applying negation to relationship, membership and subsumption queries discussed above. These queries ask if the given ABox implies that some object is not related to some feature or some object (resp. feature) does not belong to some concept, or that one concept is not included in another concept. We start with negative relationship queries.

Lemma 3. *For any consistent LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$ and for any individual names b and y,*

1. *$\mathscr{A} \models \neg(bIy)$ iff $\neg(bIy) \in \mathscr{A}$,*
2. *$\mathscr{A} \models \neg(bR_\Box y)$ iff $\neg(bR_\Box y) \in \mathscr{A}$,*
3. *$\mathscr{A} \models \neg(yR_\Diamond b)$ iff $\neg(yR_\Diamond b) \in \mathscr{A}$.*

Proof. We only prove items 1 and 2. The proof for item 3 is similar. For item 1, the right to left implication is trivial. For the left to right implication, suppose $\mathscr{A} \models \neg(bIy)$. Then, $\mathscr{A} \cup \{bIy\}$ must be inconsistent. As b and y appear in $\mathscr{A}$, no tableaux expansion rule has the term bIy in its premise. Therefore, the tableaux completion of $\mathscr{A} \cup \{bIy\}$ is $\overline{\mathscr{A}} \cup \{bIy\}$. As $\mathscr{A}$ is consistent, $\overline{\mathscr{A}}$ does not contain clash. Therefore, since $\overline{\mathscr{A}} \cup \{bIy\}$ must contain a clash, we have $\neg(bIy) \in \overline{\mathscr{A}}$. However, note that no tableaux expansion rule can add such term for individual names b, y appearing in the original ABox $\mathscr{A}$. Therefore, $\neg(bIy) \in \mathscr{A}$.

For item 2, the right to left implication is also trivial. For the left to right implication, suppose $\mathscr{A} \models \neg(bR_\Box y)$. Then, $\mathscr{A} \cup \{bR_\Box y\}$ must be inconsistent. As b and y appear in $\mathscr{A}$, the only tableaux expansion rule having term $bR_\Box y$ in its premise is adjunction rule $R_\Box$ which adds terms $\blacklozenge bIy$ and $bI\Box y$. Again, the only rules that have any of these terms in premise are I-compatibility rules that add $bR_\Box y$ to the tableaux expansion. Therefore, the tableaux completion of $\mathscr{A} \cup \{bR_\Box y\}$ is $\overline{\mathscr{A}} \cup \{bR_\Box y, \blacklozenge bIy, bI\Box y\}$. As $\mathscr{A}$ is consistent, $\overline{\mathscr{A}}$ does not contain a clash. Therefore, as $\overline{\mathscr{A}} \cup \{bR_\Box y, \blacklozenge bIy, bI\Box y\}$ must contain a clash, one of the terms $\neg(bR_\Box y)$, $\neg(\blacklozenge bIy)$ or $\neg(bI\Box y)$ must be in $\overline{\mathscr{A}}$. However, no expansion rule can add the terms of any of these forms. Furthermore, the terms of the form $\neg(\blacklozenge bIy)$ or $\neg(bI\Box y)$ cannot appear in the original ABox $\mathscr{A}$. Therefore, $\neg(bR_\Box y)$ must be in $\mathscr{A}$. □

As a result, we can answer negative relationship queries over a consistent LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$ in linear time by searching through $\mathscr{A}$.

We cannot apply a similar strategy to membership queries of the form $\neg(b : C)$ or $\neg(y :: C)$, as such terms can be implied by $\mathscr{A}$ without being present in $\overline{\mathscr{A}}$. For example, consider ABox $\mathscr{A} = \{b : C_1, \neg(b : C_1 \wedge C_2)\}$ which implies $\neg(b : C_2)$, but this term does not appear in $\overline{\mathscr{A}}$. This means that the

[5]The symbol $\vee$ in this paragraph refers to join in the first-order (meta) language, and not join of concepts in LE-$\mathscr{A}\mathscr{L}\mathscr{C}$.

model obtained by tableaux algorithm is not a universal model for these types of queries. Hence, to answer queries of the form $\neg(b:C)$ or $\neg(y::C)$, we must proceed by the usual route of adding the terms $b:C$ or $y::C$ to $\mathscr{A}$ and checking the consistency of the resulting ABox. We can also consider negative subsumption queries, i.e. queries asking whether the given ABox $\mathscr{A}$ implies one concept C_1 is not included in another concept C_2, denoted as $\neg(C_1 \sqsubseteq C_2)$. Answering this query is the same as answering if the knowledge base obtained by adding the TBox axiom $C_1 \sqsubseteq C_2$ to ABox $\mathscr{A}$ is consistent. We can answer these queries for any TBox term $C_1 \sqsubseteq C_2$, such that no sub-formula of C_1 appears in C_2, by using unraveling on $C_1 \sqsubseteq C_2$, and then applying Algorithm 1.

In this and previous sections, we have discussed answering Boolean queries of all the forms which an LE-$\mathscr{ALC}$ ABox term can take. Hence, we can combine these methodologies to answer ontology *equivalence queries* asking if two ABoxes are equivalent, i.e. $\mathscr{A}_1 \equiv \mathscr{A}_2$, by checking if every term in $\mathscr{A}_2$ is implied by $\mathscr{A}_1$ and vice versa.

3.3 Separation and differentiation queries

An important set of queries is queries asking if the given knowledge base implies (ensures) that two individuals can be differentiated from each other by a certain property. In this section, we consider some queries of this type in LE-$\mathscr{ALC}$.

Separation queries are queries of the form $S(b,d) = \exists p(bIp \wedge \neg(dIp))$ or $S(y,z) = \exists p(pIy \wedge \neg(pIz))$ for two object (resp. feature) names b, d (resp. y, z) appearing in a given ABox. These queries can be understood as asking whether two given objects or features can be separated for sure using relation I based on the given knowledge base. Note that for any LE-$\mathscr{ALC}$ ABox $\mathscr{A}$,

1. $\mathscr{A} \not\models \exists p(bIp \wedge \neg(dIp))$ iff $\mathscr{A} \cup \{\forall p(bIp \Rightarrow dIp)\}$ is consistent, and
2. $\mathscr{A} \not\models \exists p(pIy \wedge \neg(pIz))$ iff $\mathscr{A} \cup \{\forall p(pIy \Rightarrow pIz)\}$ is consistent.

Therefore, a separation query $S(b,d)$ (resp. $S(y,z)$) can be answered by checking if $\mathscr{A}$ is consistent in the extension of LE-$\mathscr{ALC}$ with the axiom $\forall p(bIp \Rightarrow dIp)$ (resp. $\forall p(pIy \Rightarrow pIz)$). To this end, we consider the expansion of the LE-$\mathscr{ALC}$ tableaux algorithm with the rules

$$SA(b,d)\ \frac{bIx}{dIx} \qquad \frac{aIy}{aIz}\ SX(y,z)\ .$$

Theorem 2. *The tableaux algorithm obtained by adding the rule $SA(b,d)$ (resp. $SX(y,z)$) to the LE-$\mathscr{ALC}$ tableaux expansion rules provides a polynomial-time sound and complete decision procedure for checking the consistency of $\mathscr{A} \cup \forall p(bIp \Rightarrow dIp)$(resp. $\mathscr{A} \cup \forall p(pIy \Rightarrow pIz)$).*

Now, we will prove the termination, soundness and completeness of tableaux algorithms for checking consistency of $\mathscr{A} \cup \{\forall p(bIp \Rightarrow dIp)\}$ and $\mathscr{A} \cup \{\forall p(pIy \Rightarrow pIz)\}$ defined in Section 3.3. We only give the proof for $\mathscr{A} \cup \forall p(bIp \Rightarrow dIp)$, the proof for $\mathscr{A} \cup \forall p(pIy \Rightarrow pIz)$ would be similar.

In this paper, we will only explain the changes that must be made to the termination, soundness, and completeness proofs of the LE-$\mathscr{ALC}$ tableaux algorithm provided in [7]. We refer to [7] for details of these proofs.

Termination. To prove termination, we prove that the following lemma proved for LE-$\mathscr{ALC}$ tableaux algorithm in [7] also holds for its extension with rule $SA(b,d)$.

Lemma 4. *[7, Lemma 1] For any individual names b, and y, and concept C added during tableau expansion of $\mathscr{A}$,*

$$\Box_{\mathscr{D}}(C) \leq \Box_{\mathscr{D}}(\mathscr{A}) + 1 \text{ and } \Diamond_{\mathscr{D}}(C) \leq \Diamond_{\mathscr{D}}(\mathscr{A}) + 1, \tag{1}$$

$$-\Diamond_{\mathscr{D}}(\mathscr{A}) - 1 \leq \Diamond_{\mathscr{D}}(b) \text{ and } \Box_{\mathscr{D}}(b) \leq \Box_{\mathscr{D}}(\mathscr{A}) + 1, \tag{2}$$

$$-\Box_{\mathscr{G}}(\mathscr{A})-1\leq\Box_{\mathscr{G}}(y)\ \textit{and}\ \Diamond_{\mathscr{G}}(y)\leq\Diamond_{\mathscr{G}}(\mathscr{A})+1 \tag{3}$$

Proof. The proof proceeds by showing that the following stronger claim holds. For any tableaux expansion $\overline{\mathscr{A}}$, obtained from $\mathscr{A}$ after any finite number of expansion steps:

1. For any term $bIy\in\overline{\mathscr{A}}$, $\Box_{\mathscr{G}}(b)-\Box_{\mathscr{G}}(y)\leq\Box_{\mathscr{G}}(\mathscr{A})+1$, and $\Diamond_{\mathscr{G}}(y)-\Diamond_{\mathscr{G}}(b)\leq\Diamond_{\mathscr{G}}(\mathscr{A})+1$.
2. For any term $bR_{\Box}y\in\overline{\mathscr{A}}$, $\Box_{\mathscr{G}}(b)+1-\Box_{\mathscr{G}}(y)\leq\Box_{\mathscr{G}}(\mathscr{A})+1$, and $\Diamond_{\mathscr{G}}(y)-\Diamond_{\mathscr{G}}(b)\leq\Diamond_{\mathscr{G}}(\mathscr{A})+1$.
3. For any term $yR_{\Diamond}b\in\overline{\mathscr{A}}$, $\Box_{\mathscr{G}}(b)-\Box_{\mathscr{G}}(y)\leq\Box_{\mathscr{G}}(\mathscr{A})+1$, and $\Diamond_{\mathscr{G}}(y)+1-\Diamond_{\mathscr{G}}(b)\leq\Diamond_{\mathscr{G}}(\mathscr{A})+1$.
4. For any term $b:C\in\overline{\mathscr{A}}$, $\Box_{\mathscr{G}}(b)+\Box_{\mathscr{G}}(C)\leq\Box_{\mathscr{G}}(\mathscr{A})+1$, and $-\Diamond_{\mathscr{G}}(b)-\Diamond_{\mathscr{G}}(C)\leq 0$.
5. For any term $y::C\in\overline{\mathscr{A}}$, $-\Box_{\mathscr{G}}(y)-\Box_{\mathscr{G}}(C)\leq 0$, and $\Diamond_{\mathscr{G}}(y)+\Diamond_{\mathscr{G}}(C)\leq\Diamond_{\mathscr{G}}(\mathscr{A})+1$.

The proof proceeds by induction of number of rules applied. The proofs for initial case (i.e. for all the terms in original ABox $\mathscr{A}$) and all the LE-$\mathscr{ALC}$ tableaux expansion rule are provided in [7, Lemma 1]. Therefore, to complete the proof we need to show that the rule $SA(b,d)$ also preserves these properties. Suppose a term dIy is added from a term bIy using rule $SA(b,d)$. In this case, by induction, we have $\Box_{\mathscr{G}}(b)-\Box_{\mathscr{G}}(y)\leq\Box_{\mathscr{G}}(\mathscr{A})+1$, and $\Diamond_{\mathscr{G}}(y)-\Diamond_{\mathscr{G}}(b)\leq\Diamond_{\mathscr{G}}(\mathscr{A})+1$. As b and d are object names appearing in $\mathscr{A}$, we have $\Box_{\mathscr{G}}(b)=\Box_{\mathscr{G}}(d)=\Diamond_{\mathscr{G}}(b)=\Diamond_{\mathscr{G}}(d)=0$. Therefore, we have $\Box_{\mathscr{G}}(d)-\Box_{\mathscr{G}}(y)\leq\Box_{\mathscr{G}}(\mathscr{A})+1$, and $\Diamond_{\mathscr{G}}(y)-\Diamond_{\mathscr{G}}(d)\leq\Diamond_{\mathscr{G}}(\mathscr{A})+1$. Hence proved. □

This lemma bounds the number of new contestant and concept names that can appear in tableaux expansion. Therefore, it implies that the number of terms that can appear in tableaux expansion are bounded by $poly(size(\mathscr{A}))$. As tableaux has no branching rules, this implies termination. (See [7] for more details).

Soundness. The soundness follows immediately from the soundness for LE-$\mathscr{ALC}$ tableaux algorithm [7, Section 4.2], and the fact that the rule $SA(b,d)$ ensures that the model obtained from completion satisfies the axiom $\forall p(bIp\Rightarrow dIp)$.

Completeness. To prove completeness, we show that the following lemma proved for LE-$\mathscr{ALC}$ [7, Lemma 5] also holds for its extension with the axiom $\forall p(bIp\Rightarrow dIp)$.

Lemma 5. *For any ABox $\mathscr{A}$, any model $M=(\mathbb{F},\cdot^{\mathscr{M}})$ of $\mathscr{A}$ can be extended to a model $M'=(\mathbb{F}',\cdot^{\mathscr{M}'})$ such that $\mathbb{F}'=(A',X',I',\{R'_{\Box}\}_{\Box\in\mathscr{G}},\{R'_{\Diamond}\}_{\Diamond\in\mathscr{F}})$, $A\subseteq A'$ and $X\subseteq X'$, and moreover for every $\Box\in\mathscr{G}$ and $\Diamond\in\mathscr{F}$:*

1. There exists $a_C\in A'$ and $x_C\in X'$ such that:

$$C^{\mathscr{M}'}=(I'^{(0)}[x_C^{\mathscr{M}'}],I'^{(1)}[a_C^{\mathscr{M}'}]),\quad a_C^{\mathscr{M}'}\in[\![C^{\mathscr{M}'}]\!],\quad x_C^{\mathscr{M}'}\in([C^{\mathscr{M}'}]), \tag{4}$$

2. For every individual b in A there exist $\Diamond b$ and $\blacklozenge b$ in A' such that:

$$I'^{(1)}[\blacklozenge b]=R'^{(1)}_{\Box}[b^{\mathscr{M}'}]\quad\textit{and}\quad I'^{(1)}[\Diamond b]=R'^{(0)}_{\Diamond}[b^{\mathscr{M}'}], \tag{5}$$

3. For every individual y in X there exist $\Box y$ and $\blacksquare y$ in X' such that:

$$I'^{(0)}[\blacksquare y]=R'^{(1)}_{\Diamond}[y^{\mathscr{M}'}]\quad\textit{and}\quad I'^{(0)}[\Box y]=R'^{(0)}_{\Box}[y^{\mathscr{M}'}]. \tag{6}$$

4. For any C, $[\![C^{\mathscr{M}}]\!]=[\![C^{\mathscr{M}'}]\!]\cap A$ and $([C^{\mathscr{M}}])=([C^{\mathscr{M}'}])\cap X$.

In [7, Section 4.3], this lemma was proved by constructing such model M'. Here, we show that if the model M additionally satisfies axiom $\forall p(bIp \Rightarrow dIp)$, then so does the model M'. This follows from the fact that the model M' is constructed in such a way that (see [7, Section 4.3] for more details) for any $b,d \in A$, a term $bI^{\mathscr{M}'}x_C$ is added for some newly added element $x_C \in X' \setminus X$ iff we have $bI^{\mathscr{M}}y$ for all $y \in C^{\mathscr{M}}$. Then, as $M \models \forall p(bIp \Rightarrow dIp)$, we also get dIy for all $y \in C^{\mathscr{M}}$ which implies $dI^{\mathscr{M}'}x_C$.

The above lemma ensures that if $\mathscr{A} \cup \{\forall p(bIp \Rightarrow dIp)\}$ has a model, then it has a model with classification of objects and features. The completeness proof then proceeds by showing that if $\mathscr{A}$ is consistent, then the model for $\mathscr{A}$ with the above properties satisfies all the terms in the completion of $\mathscr{A}$. We refer to [7, Section 4.3] for details.

Hence, we can use this expanded tableaux algorithm to answer separation queries over a given ABox $\mathscr{A}$ in polynomial time. This also allows us to answer *differentiation queries* $Dif(b,d) = S(b,d) \wedge S(d,b)$ (resp. $Dif(y,z) = S(y,z) \wedge S(z,y)$) which ask if $\mathscr{A}$ implies that b and d (resp. y and z) can be differentiated from each other by the relation I in polynomial time. We can similarly define and answer the separation and differentiation queries for the relations $R_\Box$ and $R_\Diamond$. Furthermore, we can answer *identity queries* asking if $\mathscr{A}$ implies that two individuals are not identical by checking if they can be differentiated by some relation.

The strategy used to answer separation queries can also be used to answer many other interesting types of queries. For example, we can consider separation queries which ask about separation between different relations. Consider the queries $SA(R_\Box,R_\Diamond,b) = \exists p(bR_\Box p \wedge \neg(pR_\Diamond b))$, and $SX(R_\Diamond,R_\Box,y) = \exists p(yR_\Diamond p \wedge \neg(pR_\Box y))$. These queries ask if the given ABox implies that the relations $R_\Box$ and $R_\Diamond$ are the local inverses of each other in some object b or feature y appearing in the given ABox.

Note that for any LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$,

1. $\mathscr{A} \not\models \exists p(bR_\Box p \wedge \neg(pR_\Diamond b))$ iff $\mathscr{A} \cup \{\forall p(bR_\Box p \Rightarrow pR_\Diamond b)\}$ is consistent, and
2. $\mathscr{A} \not\models \exists p(yR_\Diamond p \wedge \neg(pR_\Box y))$ iff $\mathscr{A} \cup \{\forall p(yR_\Diamond p \Rightarrow pR_\Box y)\}$ is consistent.

Therefore, a separation query $SA(R_\Box,R_\Diamond,b)$ (resp. $SX(R_\Diamond,R_\Box,y)$) can be answered by checking if $\mathscr{A}$ is consistent in the extension of LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ with the axiom $\forall p(bR_\Box p \Rightarrow pR_\Diamond b)$ (resp. $\forall p(yR_\Diamond p \Rightarrow pR_\Box y)$). To this end, we consider the expansion of the LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ tableaux algorithm with the following rules

$$SA(R_\Box,R_\Diamond,b)\ \frac{bR_\Box y}{yR_\Diamond b} \qquad \frac{yR_\Diamond b}{bR_\Box y}\ SX(R_\Diamond,R_\Box,y)\ .$$

Theorem 3. *The tableaux algorithm obtained by adding rule $SA(R_\Box,R_\Diamond,b)$ (resp. $SX(R_\Box,R_\Diamond,y)$) to LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ tableaux expansion rules provides a polynomial-time sound and complete decision procedure for checking consistency of $\mathscr{A} \cup \{\forall p(bR_\Box p \Rightarrow pR_\Diamond b)\}$(resp. $\mathscr{A} \cup \{\forall p(yR_\Diamond p \Rightarrow pR_\Box y)\}$).*

Proof. See Appendix A. □

We can similarly answer the queries of the forms $SA(R_\Box,I,b) = \exists p(bR_\Box p \wedge \neg(bIp))$, and $SX(R_\Diamond,I,y) = \exists p(yR_\Diamond p \wedge \neg(pIy))$ using tableaux algorithm expanded with the rules

$$\frac{bR_\Box y}{bIy} \quad \text{and} \quad \frac{yR_\Diamond b}{bIy}\ .$$

However, this does not apply to all separation queries on relations. For example, consider queries of the form $SA(I,R_\Box,b) = \exists p(bIp \wedge \neg(bR_\Box p))$, and $SX(I,R_\Diamond,y) = \exists p(pIy \wedge \neg(yR_\Diamond p))$. This is because the expansion of LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ tableaux algorithm with the rules

$$\frac{bIy}{bR_\Box y} \quad \text{and} \quad \frac{bIy}{yR_\Diamond b}$$

may not be terminating. We leave this as part of future work.

4 Examples

In this section, we give a toy example of an LE-$\mathscr{ALC}$ knowledge base and some queries of different types to demonstrate working of various algorithms for answering these queries discussed in the paper.

Suppose, we want to create a knowledge base to represent categorization of some movies on a streaming website which can be used to answer some queries based on them. We list the concept names and individual names for objects and features appearing in the knowledge base in the following tables:

concept name	symbol	concept name	symbol	concept name	symbol
Italian movies	IM	German movies	GM	French movies	FM
European movies	EUM	Recent movies	RM	Recent drama movies	RDM
Drama movies	DM	Famous drama movies	FDM		

object	symbol	object	symbol
All the President's Men	m_1	Spirited Away	m_2
Oppenheimer	m_3	Cinema Paradiso	m_4

feature	symbol	feature	symbol	feature	symbol
German language	f_1	French language	f_2	Based on real story	f_3
Serious plot	f_4	Released after 2015	f_5	Released after 2020	f_6

Suppose the following knowledge base $\mathscr{K}_1$ presents the information obtained by the website regarding the movies, their features and their categorization into above categories from some initial source which possibly has incomplete information.

$$\begin{aligned}\mathscr{A}_1 = \{&m_4 : IM, \neg(m_4 I x_\emptyset), x_\emptyset :: FM \wedge IM,\\ &x_\emptyset :: GM \wedge IM, f_1 :: GM, f_2 :: FM,\\ &f_4 :: DM, m_3 : RDM, m_3 I f_6, m_1 I f_3,\\ &\neg(m_1 I f_2), \neg(m_2 : EUM)\},\end{aligned} \qquad \begin{aligned}\mathscr{T}_1 = \{&EUM \equiv GM \vee FM,\\ &RDM \equiv RM \wedge DM,\\ &IM \sqsubseteq EUM\}.\end{aligned}$$

Since TBox $\mathscr{T}_1$ is acyclic, we can convert the knowledge base $\mathscr{K}_1$ into an equivalent knowledge base with only ABox using unraveling, which would be suitable for our algorithms.

For any movie m and feature y, we have mIy (resp. $\neg(mIy)$) iff according to the initial source database, movie m has (resp. does not have) feature y. The feature $x_\emptyset$ intuitively represents a contradiction. The terms $x_\emptyset :: FM \wedge IM$ and $x_\emptyset :: GM \wedge IM$ states that there is no movie that is both a French movie and Italian movie or both a German and Italian movie. The term $m_4 : IM$ specifies that Cinema Paradiso is an Italian movie. The term $f_3 :: AM$ states that Action movies have action sequences. Other terms in $\mathscr{A}_1$ can be explained similarly. The term $EUM \equiv GM \vee FM$ states that the category of European movies is the smallest category on the website which contains both German movies and French movies. The term $IM \sqsubseteq EUM$ can be equivalently written as $IM \equiv EUM \wedge C$ for some new category C, meaning all Italian movies are European movies. Other terms in $\mathscr{T}_1$ can be explained similarly. Note that the terms $\neg(m_4 I x_\emptyset)$, $x_\emptyset :: FM \wedge IM$, and $x_\emptyset :: GM \wedge IM$ together imply that m_4 is not in $(FM \wedge IM) \vee (GM \wedge IM)$. However, m_4 is in $IM = EUM \wedge IM = (GM \vee FM) \wedge IM$. Therefore, this knowledge base is inconsistent in distributive logic but it is consistent in our setting of LE-$\mathscr{ALC}$.

Additionally, the website also tries to get an understanding of subjective (epistemic) view of different user groups on the website regarding movies, their features, and categorization. To this end, website asks some users from different groups the following two questions:

(a) Given a list of movies:

(a1) Please choose movies which have feature y from the list;

(a2) please choose movies which do not have feature y from the list.

(b) Given a list of features:

(b1) please choose features that describe movie m from the list;

(b2) please choose features which do not describe movie m from the list.

Note that there can be movies (resp. features) in the list of options which are not chosen as answer to either (a1) or (a2) (resp. (b1) or (b2)).

We model information obtained from above questions as follows: If some user from group i chooses movie m (resp. movie m, resp. feature y, resp. feature y) as an answer to question (a1) (resp. (a2), resp. (b1), resp. (b2)), then we add $mR_{\Box i}y$ (resp. $\neg(mR_{\Box i}y)$, resp. $yR_{\Diamond i}m$, resp. $\neg(yR_{\Diamond i}m)$) to the knowledge base. Note that, in general, none of the terms $yR_{\Diamond i}m$, and $mR_{\Box i}y$ implies other. This is because question (a) and question (b) may be asked to different users from group i.

Then, for any category C, $[R_{\Box i}]C$ denotes the category defined by objects which are reported to have all the features in $([C])$ (description of C) by some user in group i. Thus, $[R_{\Box i}]C$ can be seen as the category of movies which are considered to be in C according to the user group i. This means that for any movie m in $[R_{\Box i}]C$ and any feature y of C, some user in the group i will name m as a movie having feature y as an answer to (a1).

Similarly, $\langle R_{\Diamond i}\rangle C$ denotes the category defined by features which all objects in $[\![C]\!]$ (objects in C) are reported to have by some user in group i. Thus, $\langle R_{\Diamond i}\rangle C$ can be seen as the category of movies defined by features which are considered to be in the description of C according to the user group i. This means that for any feature y of $\langle R_{\Diamond i}\rangle C$ and any movie m in C, some user in the group i will name y as a feature of the movie m as an answer to (b1).

Moreover, the website can also ask users the following questions:

(c) Please choose movies which belong to the category C from a given list of movies.

(d) Please choose features that describe the category C from a given list of features.

If m (resp. y) is chosen as an answer of (c) (resp. (d)) by some user in group i, then we add term $m : [R_{\Box i}]C$ (resp. $y :: \langle R_{\Diamond i}\rangle C$) to the knowledge base. Here, we assume that for any feature y in description of C, if some user chooses m as a movie in category C, then there is some user from the same group who will also choose m as a movie with feature y in answer to (a1). This assumption ensures that the $[R_{\Box i}]C$ is interpreted in accordance with LE-$\mathscr{ALC}$ semantics from relation $R_{\Box i}$.

Similarly, for any movie m in C, we assume that if some user chooses y as a feature in category C, then there is some user from the same group will also choose y as a feature with movie m in answer to (b1). This assumption ensures that the $\langle R_{\Diamond i}\rangle C$ is interpreted in accordance with LE-$\mathscr{ALC}$ semantics from relation $R_{\Diamond i}$ [6].

The following table presents knowledge base $\mathscr{K}_2$ representing different user groups' views regarding the movies obtained from the answers to the above questions. For simplictiy, we assume that we have only two different user groups. From here on, we will use $\Box_i$ (resp. $\Diamond_i$) to denote $[R_{\Box i}]$ (resp. $\langle R_{\Diamond i}\rangle$) for $i = 1, 2$.

$$\mathscr{A}_2 = \{m_3R_{\Box 1}f_3, m_3R_{\Box 2}f_3, \neg(m_1R_{\Box 1}f_6), f_3R_{\Diamond 1}m_3, f_3R_{\Diamond 2}m_3, m_3 : \Box_2 RDM, f_5 :: \Diamond_1 RM, \neg(m_1R_{\Box 2}f_5)\},$$

$$\mathscr{T}_2 = \{FDM \equiv \Box_1 DM \wedge \Box_2 DM\}.$$

[6] These assumptions can be justified if we assume we have a large number of users in each group so that at least some users in the group will have the information regarding all the movies and their features under consideration.

Let $\mathscr{K} = \mathscr{K}_1 \cup \mathscr{K}_2$ be the knowledge base obtained by combining knowledge from the source database and from users. Given the knowledge base $\mathscr{K}$, we can answer the following queries.

Positive queries. By Lemma 1, these queries can be answered using the universal model constructed using the Tableaux Algorithm 1 from the ABox obtained by unraveling $\mathscr{K}$. We depict this model in Appendix B.

(1) $q(p) = m_3Ip$ asking to name all the features implied by $\mathscr{K}$ that the movie Oppenheimer has. Using the universal model, we can give the answer $q(a) = \{f_4, f_6\}$.

(2) $q = m_4 : FDM$ asking if $\mathscr{K}$ implies that Cinema Paradiso is a Famous drama movie. We can give answer 'No' since in the universal model, $I(m_4, x_{FDM}) = 0$.

(3) $q = \Box_2 RDM \sqsubseteq \Box_2 DM$ asking if $\mathscr{K}$ implies that all the movies considered to be recent drama movies by users in group 2 are also considered to be drama movies by them. We can give answer 'Yes' since in the universal model, $a_{\Box_2 RDM} I x_{\Box_2 DM}$.

(4) $q = m_3 : \Box_2\Diamond_1 RM$ asking if $\mathscr{K}$ implies whether for any feature which is considered to be in the description of Recent movies according to some user in group 1, there is some user in group 2 who considers (reports) Oppenheimer to have this feature. We can give answer 'No' since in the universal model of the knowledge base $\mathscr{K}' = \mathscr{K} \cup \{a_{\Box_2\Diamond_1 RM} : \Box_2\Diamond_1 RM, x_{\Box_2\Diamond_1 RM} :: \Box_2\Diamond_1 RM,\ I(m_3, x_{\Box_2\Diamond_1 RM}) = 0$. In Appendix B, we provide the universal model for $\mathscr{K}$ which is obtained from complete tableaux expansion of $\mathscr{K}$. It is easy to check from the shape of the tableaux expansion rules that no tableaux expansion rule can add term $m_3 I x_{\Box_2\Diamond_1 RM}$ during the expansion of knowledge base $\mathscr{K}'$. Hence, $I(m_3, x_{\Box_2\Diamond_1 RM}) = 0$ holds in the universal model of $\mathscr{K}'$.

Negative queries. (1) $q = \neg(m_1 : \Box_2\Diamond_1 RM)$ asking if $\mathscr{K}$ implies that the movie All the President's Men is not a movie in category $\Box_2\Diamond_1 RM$ (interpretation of this category is mentioned in previous example). We can give the answer 'Yes' since, if we add the term $m_1 : \Box_2\Diamond_1 RM$ to $\mathscr{K}$, this term along with the term $f_5 :: \Diamond_1 RM$ appearing in $\mathscr{K}$, we would get $m_1 R_{\Box 2} f_5$ by rule $\Box$. Thus, the resulting knowledge base is not consistent, which means that $\mathscr{K}$ implies $\neg(m_1 : \Box_2\Diamond_1 RM)$.

(2) $q = \neg(m_3 R_{\Box 1} f_4)$ asking if $\mathscr{K}$ implies that some user in group 1 considers Oppenheimer to be a movie which does not have a serious plot. We can give the answer 'No' since the ABox obtained by unraveling $\mathscr{K}$ does not contain the term $\neg(m_3 R_{\Box 1} f_4)$ and by Lemma 3 it is not implied by $\mathscr{K}$.

Separation queries. (1) $q = Dif(m_2, m_4)$ asking if $\mathscr{K}$ implies that there is a feature that one of the movies Spirited Away and Cinema Paradiso has but the other does not. We can give the answer 'Yes'. If we add the rules $SA(m_2, m_4)$ and $SA(m_4, m_2)$ to the LE-$\mathscr{ALC}$ tableaux expansion rules and run the resulting tableaux algorithm on ABox obtained by unraveling $\mathscr{K}$, we will get the clash as showed below.

rules	**premises**	**added terms**
create		$x_{GM\vee FM} :: GM \vee FM$
$\wedge_A$	$m_4 : (GM \vee FM) \wedge C$	$m_4 : GM \vee FM,\ m_4 : C$
I	$m_4 : GM \vee FM,\ x_{GM\vee FM} :: GM \vee FM$	$m_4 I x_{GM\vee FM}$
$SA(m_4, m_2)$	$m_4 I x_{GM\vee FM}$	$m_2 I x_{GM\vee FM}$
$\neg x$	$\neg(m_2 : GM \vee FM)$	$\neg(m_2 I x_{GM\vee FM})$

(2) $q = SA(R_{\Box 1}, I, m_4)$ asking if $\mathscr{K}$ implies that there is a feature that some user in group 1 considers Cinema Paradiso has but according to the initial source database it does not. We can give the answer 'No', since if we add the rule $SA(R_{\Box 1}, I, m_4)$ to the LE-$\mathscr{ALC}$ tableaux expansion rules and run the resulting tableaux algorithm on ABox obtained by unraveling $\mathscr{K}$, we will get no clash, i.e. $\mathscr{K}$ is consistent in the extension of LE-$\mathscr{ALC}$ with the axioms $\forall y(m_4 R_{\Box 1} y \Rightarrow m_4 I y)$.

5 Conclusion and future work

In this paper, we have shown that the tableaux algorithm for LE-$\mathscr{ALC}$, or its extension with appropriate rules, can be used to answer several types of queries over LE-$\mathscr{ALC}$ ABoxes in polynomial time. Additionally, can generalize these algorithms to exponential time algorithms for LE-$\mathscr{ALC}$ knowledge bases with acyclic TBoxes by unraveling.

Dealing with cyclic TBoxes and RBox axioms. In this paper, we introduced a tableaux algorithm only for knowledge bases with acyclic TBoxes. In the future, we intend to generalize the algorithm to deal with cyclic TBoxes as well. Another interesting avenue of research is to develop tableaux and query answering algorithms for extensions of LE-$\mathscr{ALC}$ with RBox axioms. RBox axioms are used in description logics to describe the relationship between different relations in knowledge bases and the properties of these relations such as reflexivity, symmetry, and transitivity. It would be interesting to see if it is possible to obtain necessary and/or sufficient conditions on the shape of RBox axioms for which a tableaux algorithm can be obtained. This has an interesting relationship with the problem in LE-logic of providing computationally efficient proof systems for various extensions of LE-logic in a modular manner [18, 5].

Universal models for other types of queries In this work, we showed that the model constructed from tableaux Algorithm 1 acts as universal model for several types of positive queries. In the future, it would be interesting to study if we can develop tableaux algorithms in such way that the resulting models can act as universal models for negative queries and other types of queries. This would allow the algorithm to answer multiple such queries efficiently.

Answering more types of queries In Section 3.3, we mentioned that certain separation queries cannot be answered by our method due to potential non-termination of tableaux arising from the naive extension LE-$\mathscr{ALC}$ tableaux expansion rules corresponding to these queries. However, it may be possible to achieve termination in some of these case by incorporating appropriate loop check conditions into these expansion rules. In the future, we intend to study such extensions.

Generalizing to more expressive description logics. The LE-$\mathscr{ALC}$ is the non-distributive counterpart of $\mathscr{ALC}$. A natural direction for further research is to explore the non-distributive counterparts of extensions of $\mathscr{ALC}$ such as $\mathscr{ALCI}$ and $\mathscr{ALCIN}$ and fuzzy generalizations of such description logics. This would allow us to express more constructions like concepts generated by an object or a feature, which can not be expressed in LE-$\mathscr{ALC}$. This would provide us language to answer many more types of interesting queries regarding enriched formal contexts.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi & P.F. Patel-Schneider (2003): **The Description Logic Handbook: Theory, Implementation and Applications**. Cambridge University Press, doi:10.1017/CBO9780511711787.

[2] F. Baader, I. Horrocks, C. Lutz & U. Sattler (2017): **An Introduction to Description Logic**. Cambridge University Press, doi:10.1017/9781139025355.

[3] Franz Baader, Ian Horrocks & Ulrike Sattler (2005): **Description logics as ontology languages for the semantic web**. In: *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, Springer, pp. 228–248, doi:10.1007/978-3-540-32254-2_14.

[4] Daniela Berardi, Diego Calvanese & Giuseppe De Giacomo (2005): **Reasoning on UML class diagrams**. *Artificial intelligence* 168(1-2), pp. 70–118, doi:10.1016/j.artint.2005.05.003.

[5] Ineke van der Berg, Andrea De Domenico, Giuseppe Greco, Krishna B. Manoorkar, Alessandra Palmigiano & Mattia Panettiere (2023): **Labelled Calculi for the Logics of Rough Concepts**. *Logic and Its Applications*, pp. 172–188, doi:10.1007/978-3-031-26689-8_13.

[6] Ineke van der Berg, Andrea De Domenico, Giuseppe Greco, Krishna B Manoorkar, Alessandra Palmigiano & Mattia Panettiere (2023): **Non-distributive description logic**. In: *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, Springer Nature Switzerland Cham, pp. 49–69, doi:10.1007/978-3-031-43513-3_4.

[7] Ineke van der Berg, Andrea De Domenico, Giuseppe Greco, Krishna B. Manoorkar, Alessandra Palmigiano & Mattia Panettiere (2024): **Non-distributive description logic**. arXiv:2307.09561.

[8] Meghyn Bienvenu & Magdalena Ortiz (2015): **Ontology-mediated query answering with data-tractable description logics**. *Reasoning Web. Web Logic Rules: 11th International Summer School 2015, Berlin, Germany, July 31-August 4, 2015, Tutorial Lectures. 11*, pp. 218–307, doi:10.1007/978-3-319-21768-0_9.

[9] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini & Riccardo Rosati (2007): **Tractable reasoning and efficient query answering in description logics: The DL-Lite family**. *Journal of Automated reasoning* 39, pp. 385–429, doi:10.1007/s10817-007-9078-x.

[10] Willem Conradie, Sabine Frittella, Krishna Manoorkar, Sajad Nazari, Alessandra Palmigiano, Apostolos Tzimoulis & Nachoem M Wijnberg (2021): **Rough concepts**. *Information Sciences* 561, pp. 371–413, doi:10.1016/j.ins.2020.05.074.

[11] Willem Conradie, Sabine Frittella, A Palmigiano, M Piazzai, A Tzimoulis & Nachoem M Wijnberg (2017): **Toward an epistemic-logical theory of categorization**. *Electronic Proceedings in Theoretical Computer Science, EPTCS* 251, doi:10.48550/arXiv.1707.08743.

[12] Willem Conradie, Sabine Frittella, Alessandra Palmigiano, Michele Piazzai, Apostolos Tzimoulis & Nachoem M Wijnberg (2016): **Categories: how I learned to stop worrying and love two sorts**. In: *International Workshop on Logic, Language, Information, and Computation*, Springer, pp. 145–164, doi:10.48550/arXiv.1604.00777.

[13] Willem Conradie & Alessandra Palmigiano (2019): **Algorithmic correspondence and canonicity for non-distributive logics**. *Annals of Pure and Applied Logic* 170(9), pp. 923–974, doi:10.1016/j.apal.2019.04.003.

[14] Willem Conradie, Alessandra Palmigiano, Claudette Robinson & Nachoem Wijnberg (2020): **Non-distributive logics: from semantics to meaning**. In A. Rezus, editor: *Contemporary Logic and Computing*, *Landscapes in Logic* 1, College Publications, pp. 38–86, doi:10.48550/arXiv.2002.04257.

[15] Bernhard Ganter & Rudolf Wille (1997): **Applied lattice theory: Formal concept analysis**. In: *General Lattice Theory, G. Grätzer editor, Birkhäuser*, Citeseer. Available at `https://www.academia.edu/1045558/Applied_lattice_theory_Formal_concept_analysis`.

[16] Bernhard Ganter & Rudolf Wille (2012): **Formal concept analysis: mathematical foundations**. Springer Science & Business Media, doi:10.1007/978-3-642-59830-2.

[17] Birte Glimm, Carsten Lutz, Ian Horrocks & Ulrike Sattler (2008): **Conjunctive query answering for the description logic SHIQ**. *Journal of artificial intelligence research* 31, pp. 157–204, doi:10.1016/j.jcss.2011.02.012.

[18] Giuseppe Greco, Minghui Ma, Alessandra Palmigiano, Apostolos Tzimoulis & Zhiguang Zhao (2016): **Unified correspondence as a proof-theoretic tool**. *Journal of Logic and Computation* 28(7), p. 1367–1442, doi:10.1093/logcom/exw022.

[19] Roman Kontchakov & Michael Zakharyaschev (2014): **An Introduction to Description Logics and Query Rewriting**, pp. 195–244. Springer International Publishing, Cham, doi:10.1007/978-3-319-10587-1_5.

[20] Deborah L. McGuinness, Richard Fikes, James A. Hendler & Lynn Andrea Stein (2002): **DAML+OIL: An Ontology Language for the Semantic Web**. *IEEE Intell. Syst.* 17(5), pp. 72–80, doi:10.1109/MIS.2002.1039835.

[21] Steffen Staab & Rudi Studer (2010): **Handbook on ontologies**. Springer Science & Business Media, doi:10.1007/978-3-540-92673-3.

A Proof of Theorem 3

We only give the proof for $SA(R_\Box, R_\Diamond, b)$. The proof for $SX(R_\Diamond, R_\Box, y)$ is similar. The proof for soundness and completeness is analogous to the proof for soundness and completeness of Theorem 2 given in Section 3.3. To prove termination, we would need the following lemma.

Definition 1. *We define $\Diamond$-leading concepts as the smallest set of concepts satisfying the following conditions.*

1. *For any atomic concept C, the concept $\Diamond C$ is $\Diamond$-leading.*
2. *If C is $\Diamond$-leading, then $\Diamond C$ is $\Diamond$-leading.*
3. *If C is $\Diamond$-leading, then $C \vee C_1$ is $\Diamond$-leading for any C_1.*
4. *If C_1 and C_2 are $\Diamond$-leading, then $C_1 \wedge C_2$ is $\Diamond$-leading.*

We define $\Box$-leading concepts as the smallest set of concepts satisfying the following conditions.

1. *For any atomic concept C, the concept $\Box C$ is $\Box$-leading.*
2. *If C is $\Box$-leading, then $\Box C$ is $\Box$-leading.*
3. *If C is $\Box$-leading, then $C \wedge C_1$ is $\Box$-leading for any C_1.*
4. *If C_1 and C_2 are $\Box$-leading, then $C_1 \vee C_2$ is $\Box$-leading.*

Note that a concept $C_1 \wedge C_2$ (resp. $C_1 \vee C_2$) is $\Diamond$-leading (resp. $\Box$-leading) iff both C_1 and C_2 are $\Diamond$-leading (resp. $\Box$-leading).

Lemma 6. *For any LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$ and any individual names b, y, the following holds:*

1. *If a term of the form $\Diamond b I x_C$ or $\Diamond b : C$ or $x_C R_\Diamond b$ or $bI\blacksquare x_C$ appears in $\overline{\mathscr{A}}$, then C must be $\Diamond$-leading.*
2. *If a term of the form $a_C I \Box y$ or $\Box y :: C$ or $a_C R_\Box y$ or $\blacklozenge a_C I y$ appears in $\overline{\mathscr{A}}$, then C must be $\Box$-leading.*
3. *If we have a term of the form $x_C :: C'$ or $a_{C'} I x_C$ or $a_{C'} : C$ in $\overline{\mathscr{A}}$, and C is $\Box$-leading, then C' is also $\Box$-leading.*
4. *If we have a term of the form $a_C : C'$ or $a_C I x_{C'}$ or $x_{C'} :: C$ in $\overline{\mathscr{A}}$, and C is $\Diamond$-leading, then C' is also $\Diamond$-leading.*
5. *No term of the form $\Diamond b I \Box y$ can belong to $\overline{\mathscr{A}}$.*
6. *No term of the form $\Diamond b R_\Box y$ can belong to $\overline{\mathscr{A}}$.*
7. *No constant of the form $\blacklozenge\Diamond b$ or $\blacksquare\Box y$ appears in $\overline{\mathscr{A}}$ for any b or y.*

Proof. The proof follows by a simultaneous induction on the number of applications of the expansion rules. The proof for base case is obvious as $\mathscr{A}$ does not contain individual name of the form $\Diamond b$ or $\Box y$. We give the proof for all inductive cases now.

Creation rule: Only terms added by this rule are of the form $a_C : C$ or $x_C :: C$ for some $C \in \mathscr{A}$. For terms of both of these types, all the items in lemma hold trivially.

Basic rule: In this case, we add term bIy from terms $b : C$ and $y :: C$. We only need to consider the following cases: (1) b is of the form $\Diamond d$, and y is of the form $x_{C'}$ for some C'. By induction item 1, C is $\Diamond$-leading. Hence, by induction item 4, C' is also $\Diamond$-leading. Therefore, the new term $\Diamond d I x_{C'}$ also satisfies item 1. (2) b is of the form $a_{C'}$ for some C', and y is of the form $\Box z$. By induction item 2, C is $\Box$-leading. Hence, by induction item 3, C' is also $\Box$-leading. Therefore, the new term $a_{C'} I \Box z$ also satisfies item 2. (3) b is of the form a_{C_1}, and y is of the form x_{C_2}. In this case, if C_1 (resp. C_2) is $\Diamond$-leading

(resp. $\Box$-leading), then by induction item 4 (resp. item 3) C would be be $\Diamond$-leading (resp. $\Box$-leading). By again applying the same items, we would get C_2 (resp. C_1) is $\Diamond$-leading (resp. $\Box$-leading). Therefore, the added term $a_{C_1} I x_{C_2}$ satisfies items 3 and 4. Item 5 is satisfied, since if any of these terms is of the form $\Diamond bI\Box y$, then both $\Diamond b : C$, and $\Box y :: C$ appear in $\overline{\mathscr{A}}$. By induction items 1 and 2 C must be both $\Box$-leading and $\Diamond$-leading. However, no such concept exists. Item 7 is satisfied as this rule does not add new individual names.

Rules $\wedge_A$ and $\vee_X$: We only give the proof for $\wedge_A$, the proof for $\vee_X$ is dual. In this case, we add terms $b : C_1$, and $b : C_2$ from term $b : C_1 \wedge C_2$. We need to consider the following cases: (1) b is of form $\Diamond d$. By induction item 1, $C_1 \wedge C_2$ is $\Diamond$-leading. Therefore, both C_1 and C_2 must be $\Diamond$-leading. Hence, the newly added terms $b : C_1$ and $b : C_2$ also satisfy item 1. (2) b is of the form a_C. We have to show items 3 and 4 hold. If C is $\Diamond$-leading, then by induction item 4, $C_1 \wedge C_2$ is $\Diamond$-leading, which implies that both C_1 and C_2 are $\Diamond$-leading. Hence, the added terms $b : C_1$ and $b : C_2$ satisfy item 4. To show item 3 holds for the new terms, w.l.o.g. suppose C_1 is $\Box$-leading. Then, by def. $C_1 \wedge C_2$ is $\Box$-leading as well. Therefore, by induction item 3, C is $\Box$-leading. We can similarly show C is $\Box$-leading, when C_2 is $\Box$-leading. Item 7 is satisfied as this rule does not add new individual names.

Rules $\Box$ and $\Diamond$: We only give the proof for $\Box$, the proof for $\Diamond$ is dual. In this case, we add term of the form $bR_\Box y$ from terms $b : [R_\Box]C$ and $y :: C$. By induction item 6, b can not be of the form $\Diamond d$. If b is of the form $a_{C'}$ for some C', then by induction item 3, C' is $\Box$-leading. Hence, the added term $a_{C'} R_\Box y$ satisfies item 2. It also satisfies item 6 because C' being $\Box$-leading can not have $\Diamond$ as the outermost connective. Item 7 is satisfied as this rule does not add new individual names.

Rules $\Box y$, $\blacksquare y$, $\Diamond b$, and $\blacklozenge b$: We only give the proof for $\Box y$, the proofs for other rules are similar. In this case, we add term of the form $bR_\Box y$ from term $bI\Box y$. By induction item 6, b can not be of the form $\Diamond d$. If b is of the form a_C for some C, then by induction item 2, C must be $\Box$-leading. Therefore, the added term $a_C R_\Box y$ satisfies item 2. It also satisfies item 6, since C being $\Box$-leading, can not have $\Diamond$ as the outermost connective. Item 7 is satisfied as this rule does not add new individual names.

Rules $\wedge_A^{-1}$ and $\vee_X^{-1}$: We only give the proof for $\wedge_A^{-1}$, the proof for $\vee_X^{-1}$ is dual. In this case, we add term of the form $b : C_1 \wedge C_2$ from terms of the form $b : C_1$ and $b : C_2$. We need to consider the following cases: (1) b is of the form $\Diamond d$, then by induction item 1, C_1 and C_2 are $\Diamond$-leading. Then, by def. $C_1 \wedge C_2$ is also $\Diamond$-leading. Therefore, the new term $b : C_1 \wedge C_2$ satisfies item 1. (2) b is of the form a_C. We have to show items 3 and 4 hold. If C is $\Diamond$-leading, then by induction item 4, both C_1 and C_2 are $\Diamond$-leading, which implies that $C_1 \wedge C_2$ is $\Diamond$-leading. Hence, new term $b : C_1 \wedge C_2$ satisfies item 4. To show item 3 holds for the new term, suppose $C_1 \wedge C_2$ is $\Box$-leading. Then, C_1 is $\Box$-leading or C_2 is $\Box$-leading. Therefore, by induction item 3, C is $\Box$-leading. It also satisfies item 6, since C being $\Box$-leading, can not have $\Diamond$ as the outermost connective. Item 7 is satisfied as this rule does not add new individual names.

Rules $R_\Box$ and $R_\Diamond$: We only the give proof for $R_\Box$, the proof for $R_\Diamond$ is dual. In this case, we add terms $bI\Box y$, and $\blacklozenge bIy$ from $bR_\Box y$. By induction item 5, b can not be of the form $\Diamond d$. If b is of the form a_C for some C, then by induction item 2, C must be $\Box$-leading. Therefore, the added terms $a_C I\Box y$ and $\blacklozenge a_C Iy$ satisfy item 2. As b can not be of the form $\Diamond d$, the possibly new constant $\blacklozenge b$ is not of the form $\blacklozenge\Diamond d$ for any d. Hence, item 3 is satisfied.

Rules a_C and x_C: We only give the proof for a_C, the proof for x_C is dual. In this case, we add term of the form $b : C$ from term bIx_C. We need to consider the following cases: (1) b is of the form $\Diamond d$, then by induction item 1, C must be $\Diamond$-leading. Therefore, the added term $b : C$ also satisfies item 1. (2) If b is of the form $a_{C'}$ for some C'. We have to show items 3 and 4 hold. If C' is $\Diamond$-leading, then by induction item 4, C is $\Diamond$-leading. Hence, added term $a_{C'} : C$ satisfies item 4. To show item 3 holds for the new term, w.l.o.g. suppose C is $\Box$-leading. Then, by induction item 3, C' is $\Box$-leading.

$\square$

As a corollary, we get the following result.

Corollary 2. *For any LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$, and any terms $dR_{\Box}z$, and $yR_{\Diamond}b$, if $dR_{\Box}z \in \overline{\mathscr{A} \cup \{yR_{\Diamond}b\}}$, then $dR_{\Box}z \in \overline{\mathscr{A}}$.*

Proof. For any term of the form $yR_{\Diamond}b$, the only rule that has it in premise is the adjunction rule $R_{\Diamond}$ which adds terms $\Diamond bIy$, and $bI\blacksquare y$. The term $bI\blacksquare y$ cannot lead to the addition of any other term. If the term $\Diamond bIy$ leads to the addition of a term of the form $dR_{\Box}y$, then it means that we must have $\Diamond b : \Box C \in \overline{\mathscr{A}}$, for some C or $\Diamond bI\Box z \in \overline{\mathscr{A}}$ for some z. However, none of these is possible by the lemma 6. □

This lemma immediately implies the following modified version of Lemma 4.

Lemma 7. *For any individual names b, and y, and concept C added during tableau expansion of $\mathscr{A}$,*

$$\Box_{\mathscr{D}}(C) \leq \Box_{\mathscr{D}}(\mathscr{A}) + 1 \text{ and } \Diamond_{\mathscr{D}}(C) \leq \Diamond_{\mathscr{D}}(\mathscr{A}) + 1, \tag{7}$$

$$-\Diamond_{\mathscr{D}}(\mathscr{A}) - 2 \leq \Diamond_{\mathscr{D}}(b) \text{ and } \Box_{\mathscr{D}}(b) \leq \Box_{\mathscr{D}}(\mathscr{A}) + 1, \tag{8}$$

$$-\Box_{\mathscr{D}}(\mathscr{A}) - 1 \leq \Box_{\mathscr{D}}(y) \text{ and } \Diamond_{\mathscr{D}}(y) \leq \Diamond_{\mathscr{D}}(\mathscr{A}) + 2 \tag{9}$$

Proof. The proof proceeds by showing that the following stronger claim holds. For any tableaux expansion $\overline{\mathscr{A}}$, obtained from $\mathscr{A}$ after any finite number of expansion steps:

1. For any term $bIy \in \overline{\mathscr{A}}$, $\Box_{\mathscr{D}}(b) - \Box_{\mathscr{D}}(y) \leq \Box_{\mathscr{D}}(\mathscr{A}) + 1$, and $\Diamond_{\mathscr{D}}(y) - \Diamond_{\mathscr{D}}(b) \leq \Diamond_{\mathscr{D}}(\mathscr{A}) + 2$.
2. For any term $bR_{\Box}y \in \overline{\mathscr{A}}$, $\Box_{\mathscr{D}}(b) + 1 - \Box_{\mathscr{D}}(y) \leq \Box_{\mathscr{D}}(\mathscr{A}) + 1$, and $\Diamond_{\mathscr{D}}(y) - \Diamond_{\mathscr{D}}(b) \leq \Diamond_{\mathscr{D}}(\mathscr{A}) + 1$.
3. For any term $yR_{\Diamond}b \in \overline{\mathscr{A}}$, $\Box_{\mathscr{D}}(b) - \Box_{\mathscr{D}}(y) \leq \Box_{\mathscr{D}}(\mathscr{A}) + 1$, and $\Diamond_{\mathscr{D}}(y) + 1 - \Diamond_{\mathscr{D}}(b) \leq \Diamond_{\mathscr{D}}(\mathscr{A}) + 2$.
4. For any term $b : C \in \overline{\mathscr{A}}$, $\Box_{\mathscr{D}}(b) + \Box_{\mathscr{D}}(C) \leq \Box_{\mathscr{D}}(\mathscr{A}) + 1$, and $-\Diamond_{\mathscr{D}}(b) - \Diamond_{\mathscr{D}}(C) \leq 1$.
5. For any term $y :: C \in \overline{\mathscr{A}}$, $-\Box_{\mathscr{D}}(y) - \Box_{\mathscr{D}}(C) \leq 0$, and $\Diamond_{\mathscr{D}}(y) + \Diamond_{\mathscr{D}}(C) \leq \Diamond_{\mathscr{D}}(\mathscr{A}) + 1$.

The proof relies on the idea that the new rule $SA(R_{\Box}, R_{\Diamond}, b)$ introduces a new term of the form $yR_{\Diamond}b$ from a term $bR_{\Box}y$. However, by Corollary 2 the term $bR_{\Box}y$ must belong to the LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ completion of $\mathscr{A}$. Hence, it satisfies Condition 2 by Lemma 4. The proof for all other conditions follows by a straightforward generalization of the proof of [7, Lemma 1]. □

B Models for example knowledge base

The knowledge base in Section 4 is given by $\mathscr{K} = \mathscr{K}_1 \cup \mathscr{K}_2$, where the initial source database $\mathscr{K}_1 = (\mathscr{A}_1, \mathscr{T}_1)$ is given by

$$\begin{aligned}\mathscr{A}_1 = \{&m_4 : IM, \neg(m_4 I x_{\emptyset}), x_{\emptyset} :: FM \wedge IM,\\ &x_{\emptyset} :: GM \wedge IM, f_1 :: GM, f_2 :: FM,\\ &f_4 :: DM, m_3 : RDM, m_3 I f_6, m_1 I f_3,\\ &\neg(m_1 I f_2), \neg(m_2 : EUM)\},\end{aligned}$$

$$\begin{aligned}\mathscr{T}_1 = \{&EUM \equiv GM \vee FM,\\ &RDM \equiv RM \wedge DM,\\ &IM \sqsubseteq EUM\},\end{aligned}$$

while the database from users of two different groups $\mathscr{K}_2 = (\mathscr{A}_2, \mathscr{T}_2)$ is given by

$$\mathscr{A}_2 = \{m_3 R_{\Box 1} f_3, m_3 R_{\Box 2} f_3, \neg(m_1 R_{\Box 1} f_6), f_3 R_{\Diamond 1} m_3, f_3 R_{\Diamond 2} m_3, m_3 : \Box_2 RDM, f_5 :: \Diamond_1 RM, \neg(m_1 R_{\Box 2} f_5)\},$$

$$\mathscr{T}_2 = \{FDM \equiv \Box_1 DM \wedge \Box_2 DM\}.$$

By unraveling TBoxes we get the following terms:

1. $EUM \equiv GM \vee FM$
2. $RDM \equiv RM \wedge DM$
3. $IM \equiv (GM \vee FM) \wedge C$ for some C not appearing in $\mathscr{K}$
4. $FDM \equiv \Box_1 DM \wedge \Box_2 DM$

Note that the terms $FM \wedge IM$ and $GM \wedge IM$ in $\mathscr{A}_1$ are denoted as follows.

1. $FM \wedge IM \equiv FM \wedge ((GM \vee FM) \wedge C) \equiv FM \wedge C$
2. $GM \wedge IM \equiv GM \wedge ((GM \vee FM) \wedge C) \equiv GM \wedge C$

We denote the objects and features in the model of the form a_C, and x_C as below:

a_1	a_{GM}	x_1	x_{GM}
a_2	a_{FM}	x_2	x_{FM}
a_3	$a_{GM \vee FM}$	x_3	$x_{GM \vee FM}$
a_4	a_{RM}	x_4	x_{RM}
a_5	a_{DM}	x_5	x_{DM}
a_6	$a_{RM \wedge DM}$	x_6	$x_{RM \wedge DM}$
a_7	a_C	x_7	x_C
a_8	$a_{(GM \vee FM) \wedge C}$	x_8	$x_{(GM \vee FM) \wedge C}$
a_9	$a_{FM \wedge C}$	x_9	$x_{FM \wedge C}$
a_{10}	$a_{GM \wedge C}$	x_{10}	$x_{GM \wedge C}$
a_{11}	$a_{\Box_1 DM}$	x_{11}	$x_{\Box_1 DM}$
a_{12}	$a_{\Box_2 DM}$	x_{12}	$x_{\Box_2 DM}$
a_{13}	$a_{\Box_1 DM \wedge \Box_2 DM}$	x_{13}	$x_{\Box_1 DM \wedge \Box_2 DM}$
a_{14}	$a_{\Box_2 (RM \wedge DM)}$	x_{14}	$x_{\Box_2 (RM \wedge DM)}$
a_{15}	$a_{\Diamond_1 RM}$	x_{15}	$x_{\Diamond_1 RM}$
a_{16}	$a_\top$	x_{16}	$x_\bot$
		x_{17}	$x_\emptyset$

We give the following table depicting all the objects and features appearing in the model and whether or not they are related by I.

The relations $R_{\Box i}$, and $R_{\Diamond i}$ are given as follows. We have $R_{\Box 1} = \{(a_{11}, x_5), (a_{13}, x_5), (m_3, f_3))\}$, and $R_{\Box 2} = \{(a_{12}, x_5), (a_{13}, x_5), (a_{14}, x_4), (a_{14}, x_5), (a_{14}, x_6), (a_{14}, f_4), (m_3, x_4), (m_3, x_5), (m_3, x_6), (m_3, f_3), (m_3, f_4)\}$. $R_{\Diamond 1} = \{(f_5, a_4), (f_5, a_6), (f_3, m_3), (f_5, m_3), (x_{15}, a_6), (x_{15}, a_4), (x_{15}, m_3)\}$ and $R_{\Diamond 2} = \{(f_3, m_3)\}$. The model contains atomic concepts GM, FM, RM, DM, and C. For any of these concepts D, its interpretation is given by the tuple $(x_D^{\downarrow}, a_D^{\uparrow})$.

Table 1: Objects (A) and features (X) of the model and Relation I between them

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	$\blacksquare_1 x_{15}$	$\square_2 x_4$
a_1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a_2	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a_3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
a_5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
a_6	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0
a_7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
a_8	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
a_9	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
a_{10}	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0
a_{11}	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
a_{12}	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
a_{13}	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0
a_{14}	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1
a_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
a_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_1 a_{11}$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_2 a_{12}$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_1 a_{13}$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_2 a_{13}$	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_2 a_{14}$	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
$\diamond_1 a_6$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
m_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
m_3	0	0	0	1	1	1	0	0	0	0	0	1	0	1	0	0	1	1
$\blacklozenge_1 m_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_2 m_3$	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
$\diamond_1 m_3$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
$\diamond_2 m_3$	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
m_4	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

	f_1	f_2	f_3	f_4	f_5	f_6	$\blacksquare_1 f_3$	$\blacksquare_2 f_3$	$\square_1 f_3$	$\square_2 f_3$	$\blacksquare_1 f_5$	$\square_2 f_4$	x_{17}
a_1	1	0	0	0	0	0	0	0	0	0	0	0	0
a_2	0	1	0	0	0	0	0	0	0	0	0	0	0
a_3	0	0	0	0	0	0	0	0	0	0	0	0	0
a_4	0	0	0	0	0	0	0	0	0	0	1	0	0
a_5	0	0	0	1	0	0	0	0	0	0	0	0	0
a_6	0	0	0	1	0	0	0	0	0	0	1	0	0
a_7	0	0	0	0	0	0	0	0	0	0	0	0	0
a_8	0	0	0	0	0	0	0	0	0	0	0	0	0
a_9	0	0	0	0	0	0	0	0	0	0	0	0	1
a_{10}	0	0	0	0	0	0	0	0	0	0	0	0	1
a_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0
a_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0
a_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0
a_{14}	0	0	0	0	0	0	0	0	0	0	0	1	0
a_{15}	0	0	0	0	1	0	0	0	0	0	0	0	0
a_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_1 a_{11}$	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_2 a_{12}$	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_1 a_{13}$	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_2 a_{13}$	0	0	0	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_2 a_{14}$	0	0	0	1	0	0	0	0	0	0	0	0	0
$\diamond_1 a_6$	0	0	0	0	1	0	0	0	0	0	0	0	0
m_1	0	0	1	0	0	0	0	0	0	0	0	0	0
m_2	0	0	0	0	0	0	0	0	0	0	0	0	0
m_3	0	0	0	1	0	1	1	1	1	1	1	1	0
$\blacklozenge_1 m_3$	0	0	1	0	0	0	0	0	0	0	0	0	0
$\blacklozenge_2 m_3$	0	0	1	1	0	0	0	0	0	0	0	0	0
$\diamond_1 m_3$	0	0	1	0	1	0	0	0	0	0	0	0	0
$\diamond_2 m_3$	0	0	1	0	0	0	0	0	0	0	0	0	0
m_4	0	0	0	0	0	0	0	0	0	0	0	0	0

Fuzzy Lattice-based Description Logic

Yiwen Ding Krishna Manoorkar
School of Business and Economics
Vrije Universiteit Amsterdam*
Amsterdam, Netherlands
dyiwen666@gmail.com krishna.manoorkar@gmail.com

Recently, description logic LE-$\mathscr{ALC}$ was introduced for reasoning in the semantic environment of enriched formal contexts, and a polynomial-time tableaux algorithm was developed to check the consistency of knowledge bases with acyclic TBoxes [8]. In this work, we introduce a fuzzy generalization of LE-$\mathscr{ALC}$ called LE-$\mathscr{FALC}$ which provides description logic counterpart of many-valued normal non-distributive logic a.k.a. many-valued LE-logic. This description logic can be used to represent and reason about knowledge in formal framework of fuzzy formal contexts and fuzzy formal concepts as introduced in [5]. We provide a tableaux algorithm that provides a complete and sound polynomial-time decision procedure to check the consistency of LE-$\mathscr{FALC}$ ABoxes. As a result, we also obtain an exponential-time decision procedure for checking the consistency of LE-$\mathscr{FALC}$ with acyclic TBoxes by unraveling.

1 Introduction

Description Logic (DL) [2] is a class of logical formalisms, typically based on the classical first-order logic, widely used in Knowledge Representation and Reasoning to describe and reason about relevant concepts and their relationships in a given application domain.

Normal non-distributive modal logic a.k.a. LE-logic has been studied as a logic of categorization endowed with modal operators based on the fact that non-distributive modal logic is sound and complete w.r.t. its semantics based on *enriched formal contexts* (i.e. , relational structures based on formal contexts from Formal Concept Analysis (FCA)) [10, 11]. An enriched formal context dually corresponds to the concept lattice of its underlying formal context expanded with normal modal operators defined by those enriching relations. Similarly to the classical modal logic, these normal modal operators have many different intuitive interpretations, such as epistemic interpretation [11], approximation interpretation [9], etc.

In [8], the two-sorted non-distributive description logic LE-$\mathscr{ALC}$[1] was developed, based on LE-logics and their semantics based on enriched formal contexts. LE-$\mathscr{ALC}$ provides a natural means of reasoning about the formal concepts (or categories) arising from formal contexts in FCA [14, 15] enriched with modal operators. LE-$\mathscr{ALC}$ has the same relationship with non-distributive modal logic and its semantics based on formal contexts, as $\mathscr{ALC}$ with the classical normal modal logic and its Kripke frames semantics. Namely, LE-$\mathscr{ALC}$ facilitates the description of *enriched formal contexts*, and *formal concepts* generated by them. As enriched formal contexts dually correspond to complete lattices equipped with normal modal operators, LE-$\mathscr{ALC}$ is also a natural framework for representing

*This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101007627. The research of Krishna Manoorkar is supported by the NWO grant KIVI.2019.001. Yiwen Ding is supported by the China Scholarship Council.

[1]Even though concept names in LE-$\mathscr{ALC}$ do not contain negation, we still refer to this description logic as LE-$\mathscr{ALC}$ rather than LE-$\mathscr{ALE}$, as negation on ABox terms is included in the description logic language.

Cynthia Kop and Helida Salles Santos (Eds); 19th Workshop on
Logical and Semantic Frameworks with Applications (LSFA'24)
EPTCS 421, 2025, pp. 44–63, doi:10.4204/EPTCS.421.3

and reasoning about general (possibly non-distributive) complete lattices equipped with normal modal operators.

For many real-life applications, the concepts under consideration are *imprecise* or *fuzzy*. For example, if we want to categorize movies, the concepts involved such as "action movies", "dramas", etc. are imprecise. To model such scenarios, different fuzzy generalizations of FCA have been studied extensively [5, 19, 20]. In [12] the many-valued LE-logic was described as the logic of vague categorization. The semantics of this logic is given by many-valued formal contexts enriched with relations defining modal operators. The propositional part of this logic corresponds to the logic of fuzzy concepts defined in [5]. The modal operators can be given different interpretations, such as epistemic interpretation or approximation interpretation as in the crisp case [12]. In this work, we generalize the description logic LE-$\mathscr{ALC}$ to the fuzzy description logic LE-$\mathscr{FALC}$. This logic has the same relationship with the many-valued LE-logic as LE-$\mathscr{ALC}$ with (crisp) LE-logic. LE-$\mathscr{FALC}$ facilitates the description of many-valued enriched formal contexts, which give rise to fuzzy concept lattices extended with normal modal operators. Hence, LE-$\mathscr{FALC}$ is also a natural framework to represent and reason about general fuzzy FCA expanded with modal operators.

In [8], a polynomial-time tableaux algorithm for checking the consistency of LE-$\mathscr{ALC}$ ABoxes was developed. The polynomial-time upper bound is based on the following two features of LE-$\mathscr{ALC}$: (1) In the semantics of LE-$\mathscr{ALC}$ disjunction is interpreted in terms of the *intersection* of intensions. Therefore, $\vee$ does not produce branching for the same reason that $\wedge$ does not in the classical setting. (2) Unlike the classical logic, the $\Diamond$ operator in LE-$\mathscr{ALC}$ is interpreted using a universal quantifier (instead of an existential quantifier), which allows us to bound the number of new constants appearing in the tableaux expansion.

In this work, we generalize the tableaux expansion rules for LE-$\mathscr{ALC}$ to the fuzzy setting to define tableaux expansion rules for LE-$\mathscr{FALC}$. We show that the resulting tableaux algorithm provides a sound and complete polynomial-time decision procedure for checking the consistency of LE-$\mathscr{FALC}$ knowledge bases with acyclic TBoxes. Since many description logic reasoning tasks can be equivalently represented as a problem of checking the consistency of knowledge bases (cf. [2]), this tableaux algorithm provides us with an efficient methodology to perform these tasks for LE-$\mathscr{FALC}$ in polynomial time.

Structure of the paper. In Section 2, we present the required preliminaries used in this paper. In Section 3, we introduce the description logic LE-$\mathscr{FALC}$. In Section 5, we introduce the Tableaux algorithm for checking the consistency of LE-$\mathscr{FALC}$ ABoxes, which can also be easily extended to LE-$\mathscr{FALC}$ acyclic TBoxes. In Section 6, we show the soundness of this tableaux algorithm. In Section 7, we show the completeness of this tableaux algorithm. In Section 4, we give some examples of LE-$\mathscr{FALC}$ ABoxes and use our tableaux algorithm to check their consistency.

2 Preliminaries

In this section, we gather some useful facts about the many-valued LE-logic and its many-valued polarity-based semantics [12], and description logic LE-$\mathscr{ALC}$ [8]. We assume that the readers are familiar with the basic concepts from description logic, which we refer to [3]. For more details on LE-logic and its polarity-based semantics, we refer to [11], [9], and [13].

2.1 Many-valued polarity-based semantics

Let Prop be a countable set of propositional variables. The language of many-valued LE-logic $\mathscr{L}$ is defined as follows:

$$\varphi ::= p \mid \bot \mid \top \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi,$$

where $p \in$ Prop, and $\Box \in \mathscr{G}$ and $\Diamond \in \mathscr{F}$ for finite sets $\mathscr{G}$ and $\mathscr{F}$ of unary $\Box$-type (resp. $\Diamond$-type) modal operators.

In this paper, we let $\mathbf{H} = (H, \vee, \wedge, \to, 1, 0)$ be a complete and completely distributive Heyting algebra.[2]For any α, β in $\mathbf{H}$, let $\alpha \leftrightarrow \beta := \alpha \to \beta \wedge \beta \to \alpha$. For any non-empty set W, an $\mathbf{H}$-*subset* of W is a map $u : W \to \mathbf{H}$. We let $\mathbf{H}^W$ denote the set of all $\mathbf{H}$-subsets. Clearly, $\mathbf{H}^W$ induces a complete and completely distributive Heyting algebra by defining the operations pointwise. For every $\alpha \in \mathbf{H}$ and $w \in W$, $\{\alpha/w\} : W \to \mathbf{H}$ is the map defined by $w' \mapsto \alpha$ if $w' = w$ and $w' \mapsto 0$ if $w' \neq w$. Let $u, v : W \to \mathbf{H}$ be any $\mathbf{H}$-subsets, we define $u \subseteq v$ if $u(w) \leq v(w)$ for each $w \in W$, which defines a partial order on $\mathbf{H}^W$. An $\mathbf{H}$-*relation* is a map $R : U \times W \to \mathbf{H}$, where U and W are non-empty sets. Any $\mathbf{H}$-relation $R : U \times W \to \mathbf{H}$ induces maps $R^{(0)}[-] : \mathbf{H}^W \to \mathbf{H}^U$ and $R^{(1)}[-] : \mathbf{H}^U \to \mathbf{H}^W$ which are defined as follows: for every $f : U \to \mathbf{H}$, and $u : W \to \mathbf{H}$, $R^{(1)}[f] : W \to \mathbf{H}$ such that $x \mapsto \bigwedge_{a \in U}(f(a) \to R(a,x))$, and $R^{(0)}[u] : U \to \mathbf{H}$ such that $a \mapsto \bigwedge_{x \in W}(u(x) \to R(a,x))$.

Definition 1. *An* $\mathbf{H}$-valued formal context *is a tuple* $\mathfrak{P} = (A, X, I)$ *such that* A *and* X *are non-empty sets, and* $I : A \times X \to \mathbf{H}$ *is an* $\mathbf{H}$-*relation. Any* $\mathbf{H}$-*valued formal context induces maps* $(\cdot)^{\uparrow} : \mathbf{H}^A \to \mathbf{H}^X$ *and* $(\cdot)^{\downarrow} : \mathbf{H}^X \to \mathbf{H}^A$ *given by* $(\cdot)^{\uparrow} = I^{(1)}[\cdot]$ *and* $(\cdot)^{\downarrow} = I^{(0)}[\cdot]$.

Given an $\mathbf{H}$-valued formal context (A, X, I), it is easy to see that for any $f \in \mathbf{H}^A$ and $u \in \mathbf{H}^X$, there is $f \subseteq u^{\downarrow}$ iff $u \subseteq f^{\uparrow}$, which means the pair of maps $(\cdot)^{\uparrow}$ and $(\cdot)^{\downarrow}$ form a Galois connection between $(\mathbf{H}^A, \subseteq)$ and $(\mathbf{H}^X, \subseteq)$. A *fuzzy formal concept* of $\mathfrak{P}$ is a pair of maps $(f, u) \in \mathbf{H}^A \times \mathbf{H}^X$ such that $f^{\uparrow} = u$ and $u^{\downarrow} = f$. It follows immediately that if a pair of maps $(f, u) \in \mathbf{H}^A \times \mathbf{H}^X$ is a fuzzy formal concept, then there is $f^{\uparrow\downarrow} = f$ and $u^{\downarrow\uparrow} = u$, which means that f and u are *Galois-stable*. In this paper, we may use pair $([\![c]\!], ([c]))$ to denote a fuzzy concept c from a given fuzzy formal context where $[\![c]\!] \in \mathbf{H}^A$ and $([c]) \in \mathbf{H}^X$, except Section 2.2 and Section 2.3. $[\![c]\!]$ (resp. $([c])$) is the *extension* (resp. *intension*) of the fuzzy concept. The set of all fuzzy formal concepts of $\mathfrak{P}$ can be partially ordered as follows:

$$(f, u) \leq (g, v) \quad \text{iff} \quad f \subseteq g \quad \text{iff} \quad v \subseteq u.$$

Ordered in this way, the set of fuzzy formal concepts of $\mathfrak{P}$ forms a complete lattice, which we refer to as the *concept lattice* of $\mathfrak{P}$ and denote by $\mathfrak{P}^+$, such that for set K of fuzzy formal concepts of $\mathfrak{P}$, $\bigwedge \mathrm{K} := (\bigwedge_{c \in \mathrm{K}} [\![c]\!], (\bigwedge_{c \in \mathrm{K}} [\![c]\!])^{\uparrow})$ and $\bigvee \mathrm{K} := (\bigwedge_{c \in \mathrm{K}} ([c]))^{\downarrow}, \bigwedge_{c \in \mathrm{K}} ([c]))$.

Definition 2. *An* $\mathbf{H}$-valued enriched formal context *is a tuple* $\mathfrak{F} = (\mathfrak{P}, \mathscr{R}_{\Box}, \mathscr{R}_{\Diamond})$ *such that* $\mathfrak{P} = (A, X, I)$ *is an* $\mathbf{H}$-*valued formal context, and* $\mathscr{R}_{\Box} = \{R_{\Box} : A \times X \to \mathbf{H} \mid \Box \in \mathscr{G}\}$ *and* $\mathscr{R}_{\Diamond} = \{R_{\Diamond} : X \times A \to \mathbf{H} \mid \Diamond \in \mathscr{F}\}$ *are sets of* I-compatible $\mathbf{H}$-*relations, that is, for any* $\Box \in \mathscr{G}$ *and* $\Diamond \in \mathscr{F}$, $a \in A$ *and* $x \in X$, $\alpha \in \mathbf{H}$, *the* $\mathbf{H}$-*subsets* $R_{\Box}^{(0)}[\{\alpha/x\}]$, $R_{\Box}^{(1)}[\{\alpha/a\}]$, $R_{\Diamond}^{(0)}[\{\alpha/a\}]$ *and* $R_{\Diamond}^{(1)}[\{\alpha/x\}]$ *are Galois-stable.*

Every $\mathbf{H}$-valued formal context can be seen as a many-value formal context, where elements in A are "objects" and elements in X are "features". For any $a \in A$, $x \in X$ and $\alpha \in \mathbf{H}$, $I(a,x) = \alpha$ is read as "object a has feature x to degree α". The I-compatibility conditions can be understood in such way: the sets of all objects (resp. features) relating to a feature (resp. object) by modal relations $R_{\Box}$ and $R_{\Diamond}$ form concepts. These modal relations have different interpretations like epistemic interpretation [11] or approximation interpretation [9].

[2]We chose Heyting algebra here in order to maintain the simplicity and readability of the article. In fact, the results of this paper can be generalized to any complete frame-distributive and dually frame-distributive, commutative, and associative residuated lattice.(cf. [12])

Definition 3. *Let* $\mathfrak{F} = (\mathfrak{P}, \mathscr{R}_\Box, \mathscr{R}_\Diamond)$ *be any* **H***-valued enriched formal context.* $\mathfrak{F}^+ = (\mathfrak{P}^+, \{[R_\Box]\}_{\Box\in\mathscr{G}}, \{\langle R_\Diamond\rangle\}_{\Diamond\in\mathscr{F}})$ *is the* complex algebra *of* $\mathfrak{F}$*, where* $\mathfrak{P}^+$ *is the concept lattice of* $\mathfrak{P}$*, and for any* $\Box \in \mathscr{G}$ *and* $\Diamond \in \mathscr{F}$*,* $[R_\Box], \langle R_\Diamond\rangle : \mathfrak{P}^+ \to \mathfrak{P}^+$ *are maps such that for every* $c = (\llbracket c\rrbracket, (\![c]\!)) \in \mathfrak{P}^+$*,* $[R_\Box](c) := (R_\Box^{(0)}[(\![c]\!)], (R_\Box^{(0)}[(\![c]\!)])^\uparrow)$ *and* $\langle R_\Diamond\rangle(c) := ((R_\Diamond^{(0)}[\llbracket c\rrbracket])^\downarrow, R_\Diamond^{(0)}[\llbracket c\rrbracket])$.

An example of **H**-valued enriched formal context and its complex algebra is provided below.

Example 1. *Consider the Heyting algebra* **H** *which is a three-valued chain* $(\{0, 1/2, 1\}, \leq)$*. Note that, on this algebra, the operation* $\to$ *is given by* $a \to b = 1$ *if* $a \leq b$*, and* $a \to b = b$ *otherwise. Let* (A, X, I) *be the fuzzy formal context where* $A = \{a_1, a_2\}$*,* $X = \{x_1, x_2, x_3\}$*, with incidence relation* $I : A \times X \to \mathbf{H}$ *enriched with* I-compatible *fuzzy relations* $R_\Box : A \times X \to \mathbf{H}$ *and* $R_\Diamond : X \times A \to \mathbf{H}$ *given in the following tables:*

I	a_1	a_2
x_1	1	1/2
x_2	0	1
x_3	1	1/2

$R_\Box$	a_1	a_2
x_1	1	1
x_2	0	1
x_3	1	1

$R_\Diamond$	x_1	x_2	x_3
a_1	1	0	1
a_2	1	1	1

It is easy to check that the only Galois-stable fuzzy subsets of A (extents of the concepts generated) are the following:

	a_1	a_2
f_1	1	1
f_2	1	1/2
f_3	0	1
f_4	0	1/2

Therefore, the fuzzy sets $g_i = f_i^\uparrow$ *(intents of the concepts generated) are given as follows:*

	x_1	x_2	x_3
g_1	1/2	0	1/2
g_2	1	0	1
g_3	1/2	1	1/2
g_4	1	1	1

The concept lattice of (A, X, I) *is represented by the Hasse diagram 1. The operations* $[R_\Box]$ *and* $\langle R_\Diamond\rangle$ *on the lattice are given as follows:*

	(f_1, g_1)	(f_2, g_2)	(f_3, g_3)	(f_4, g_4)
$[R_\Box]$	(f_1, g_1)	(f_1, g_1)	(f_3, g_3)	(f_3, g_3)
$\langle R_\Diamond\rangle$	(f_2, g_2)	(f_2, g_2)	(f_4, g_4)	(f_4, g_4)

Definition 4. *An* **H**-model *is a tuple* $\mathfrak{M} = (\mathfrak{F}, V)$ *such that* $\mathfrak{F}$ *is an* **H***-valued enriched formal context, and* $V : \mathsf{Prop} \to \mathfrak{F}^+$ *is called an* assignment *on* $\mathfrak{F}$*. For any* $p \in \mathsf{Prop}$*, let* $V(p) := (\llbracket p\rrbracket, (\![p]\!))$*, where* $\llbracket p\rrbracket : A \to \mathbf{H}$ *and* $(\![p]\!) : X \to \mathbf{H}$ *such that* $\llbracket p\rrbracket^\uparrow = (\![p]\!)$ *and* $(\![p]\!)^\downarrow = \llbracket p\rrbracket$*.* V *can be homomorphically extended to a unique* valuation $\overline{V} : \mathscr{L} \to \mathfrak{F}^+$.

Definition 5. *Given any* **H***-model* $\mathfrak{M} = (\mathfrak{F}, V)$ *and* $\alpha \in \mathbf{H}$*, let* $1^{\mathbf{H}^A} : A \to \mathbf{H}$ *be the constant map such that* $a \mapsto 1$ *for any* $a \in A$*, and* $1^{\mathbf{H}^X} : X \to \mathbf{H}$ *be the constant map such that* $x \mapsto 1$ *for any* $x \in X$*. There are two* modal satisfaction relations $\Vdash^\alpha$ *and* $\succ^\alpha$ *defined inductively as follows:*

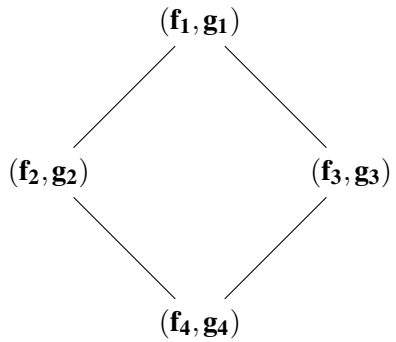


Figure 1: The concept lattice of (A, X, I)

$$
\begin{array}{rll}
\mathfrak{M}, a \Vdash^{\alpha} p & \textit{iff} & \alpha \leq [\![p]\!](a); \\
\mathfrak{M}, a \Vdash^{\alpha} \top & \textit{iff} & \alpha \leq (1^{\mathbf{H}^A})(a) \textit{ i.e. always;} \\
\mathfrak{M}, a \Vdash^{\alpha} \bot & \textit{iff} & \alpha \leq (1^{\mathbf{H}^X})^{\downarrow}(a) = \bigwedge_{x \in X} (1^{\mathbf{H}^X}(x) \to I(a,x)) = \bigwedge_{x \in X} I(a,x); \\
\mathfrak{M}, a \Vdash^{\alpha} \varphi \wedge \psi & \textit{iff} & \mathfrak{M}, a \Vdash^{\alpha} \varphi \quad \textit{and} \quad \mathfrak{M}, a \Vdash^{\alpha} \psi; \\
\mathfrak{M}, a \Vdash^{\alpha} \varphi \vee \psi & \textit{iff} & \alpha \leq (([\![\varphi]\!]) \wedge ([\![\psi]\!]))^{\downarrow}(a) = \bigwedge_{x \in X} (([\![\varphi]\!])(x) \wedge ([\![\psi]\!])(x) \to I(a,x)); \\
\mathfrak{M}, a \Vdash^{\alpha} \Box\varphi & \textit{iff} & \alpha \leq (R_{\Box}^{(0)}[([\![\varphi]\!])])(a) = \bigwedge_{x \in X} (([\![\varphi]\!])(x) \to R_{\Box}(a,x)); \\
\mathfrak{M}, a \Vdash^{\alpha} \Diamond\varphi & \textit{iff} & \alpha \leq ((R_{\Diamond}^{(0)}[[\![\varphi]\!]])^{\downarrow})(a) = \bigwedge_{x \in X} ((R_{\Diamond}^{(0)}[[\![\varphi]\!]])(x) \to I(a,x)) \\
\mathfrak{M}, x \succ^{\alpha} p & \textit{iff} & \alpha \leq ([p])(x); \\
\mathfrak{M}, x \succ^{\alpha} \bot & \textit{iff} & \alpha \leq (1^{\mathbf{H}^X})(x) \textit{ i.e. always;} \\
\mathfrak{M}, x \succ^{\alpha} \top & \textit{iff} & \alpha \leq (1^{\mathbf{H}^A})^{\uparrow}(x) = \bigwedge_{a \in A} (1^{\mathbf{H}^A}(a) \to I(a,x)) = \bigwedge_{a \in A} I(a,x); \\
\mathfrak{M}, x \succ^{\alpha} \varphi \vee \psi & \textit{iff} & \mathfrak{M}, x \succ^{\alpha} \varphi \quad \textit{and} \quad \mathfrak{M}, x \succ^{\alpha} \psi; \\
\mathfrak{M}, x \succ^{\alpha} \varphi \wedge \psi & \textit{iff} & \alpha \leq ([\![\varphi]\!] \wedge [\![\psi]\!])^{\uparrow}(x) = \bigwedge_{a \in A} ([\![\varphi]\!](a) \wedge [\![\psi]\!](a) \to I(a,x)); \\
\mathfrak{M}, x \succ^{\alpha} \Diamond\varphi & \textit{iff} & \alpha \leq (R_{\Diamond}^{(0)}[[\![\varphi]\!]])(x) = \bigwedge_{a \in A} ([\![\varphi]\!](a) \to R_{\Diamond}(x,a)), \textit{ for any } \Diamond \in \mathscr{F}; \\
\mathfrak{M}, x \succ^{\alpha} \Box\varphi & \textit{iff} & \alpha \leq ((R_{\Box}^{(0)}[([\![\varphi]\!])])^{\uparrow})(x) = \bigwedge_{a \in A} ((R_{\Box}^{(0)}[([\![\varphi]\!])])(a) \to I(a,x)), \textit{ for any } \Box \in \mathscr{G}.
\end{array}
$$

From the above definition, it is easy to check that for every $\varphi \in \mathscr{L}$, there is $\mathfrak{M}, a \Vdash^{\alpha} \varphi$ iff $\alpha \leq [\![\varphi]\!](a)$, and $\mathfrak{M}, x \succ^{\alpha} \varphi$ iff $\alpha \leq ([\varphi])(x)$. Therefore, given any $\mathbf{H}$-model $\mathfrak{M}$, a, x in $\mathfrak{M}$, $\alpha \in \mathbf{H}$ and $\varphi \in \mathscr{L}$, $\mathfrak{M}, a \Vdash^{\alpha} \varphi$ and $\mathfrak{M}, x \succ^{\alpha} \varphi$ can be read as "object a is a member of category φ to degree α" and "feature x describes category φ to degree α", respectively. Here, the interpretation of the propositional connectives $\vee$ and $\wedge$ in the framework described above reproduces the standard notion of join and the meet of fuzzy formal concepts used in fuzzy FCA. The interpretations of the modal operators $\Box$ and $\Diamond$ are motivated by algebraic properties and duality theory for modal operators on lattices.

The definitions of α-membership and α-description given above stand in the same relationship to their crisp counterparts in [10, 11] as Fitting's definition of the α-satisfaction of modal formulas on many-valued Kripke frame stands to the definition of satisfaction of modal formulas on (crisp) Kripke frames. More precisely, Fitting derives many-valued semantics for modal logic by reading the standard first-order satisfaction clauses for modal formulas as formulas of many-valued predicate logic interpreted on many-valued first-order structures in the standard way (see e.g. [17, Chapter 5]). We proceed similarly. For example, consider the defining clause for the satisfaction of a box-formula on a (crisp) enriched formal context rewritten in first-order syntax as

$$\mathfrak{M}, a \Vdash \Box\varphi \quad \text{iff} \quad \forall x(x \in ([\varphi]) \to aR_{\Box}x). \tag{1}$$

Reading the clause above as a statement of (two-sorted) **H**-valued predicate logic, the universal quantifier is interpreted as a conjunction indexed by the set X (itself interpreted as a meet in **H**), the membership statement $x \in ([\varphi])$ is now **H**-valued and more naturally written as $([\varphi])(x)$, and the same holds for the atomic formula $aR_\Box x$, which we write as $R_\Box(a,x)$. The implication $\to$ is interpreted as the implication $\to^{\mathbf{H}}$ of **H**. Furthermore, $\Vdash$ is now interpreted as an **H**-valued relation between H and $\mathscr{L}$ and therefore rather than asking if it holds between an object a and a formula, we ask whether it gives a value of at least $\alpha \in \mathbf{H}$ when applied to an object and formula. Thus, (1) is transformed into

$$\mathfrak{M}, a \Vdash^{\alpha} \Box\varphi \quad \text{iff} \quad \alpha \leq \bigwedge_{x \in X} (([\varphi])(a) \to^{\mathbf{A}} R_\Box(a,x)). \tag{2}$$

Different interpretations of modal operators in LE-logic transfers naturally to the many-valued interpretation. Instead of attributing features to objects absolutely, in this setting, agents can make such attributions in a graded way, and accordingly, their perceived categories consist of stable pairs of **H**-valued sets of object and features.

2.2 Non-distributive description logic LE-$\mathscr{A}\mathscr{L}\mathscr{C}$

The language of LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ is intended to be interpreted on the complex algebras of enriched formal contexts. The set of the *individual names* for LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ is the union of two disjoint sets OBJ and FEAT, which are interpreted as the *objects* and *features* of the enriched formal contexts, respectively. The set $\mathscr{R}$ of the *role names* for LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ is the union of three types of relations: (1) A unique relation $I \subseteq \mathsf{OBJ} \times \mathsf{FEAT}$; (2) a set of relations $\mathscr{R}_\Box$ of the form $R_\Box \subseteq \mathsf{OBJ} \times \mathsf{FEAT}$; (3) a set of relations $\mathscr{R}_\Diamond$ of the form $R_\Diamond \subseteq \mathsf{FEAT} \times \mathsf{OBJ}$. While I is intended to be interpreted as the incidence relation of enriched formal contexts and encodes information on which objects have which features, the relations in $\mathscr{R}_\Box$ and $\mathscr{R}_\Diamond$ encode additional relationships between objects and features (cf. [9] for an extended discussion). For any set $\mathscr{C}$ of *primitive concepts*, the LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ *concepts* are defined as follows:

$$C := D \mid C_1 \wedge C_2 \mid C_1 \vee C_2 \mid \langle R_\Diamond \rangle C \mid [R_\Box] C$$

where $D \in \mathscr{C}$, $R_\Box \in \mathscr{R}_\Box$ and $R_\Diamond \in \mathscr{R}_\Diamond$. Concepts such as $C_1 \vee C_2$ (resp. $C_1 \wedge C_2$) are interpreted as the smallest common superconcept (resp. the greatest common subconcept) as in FCA. Because there is no canonical and natural way to interpret negations in non-distributive (lattice-based) settings, we do not include $\neg C$ as a concept. Concepts $\langle R_\Diamond \rangle C$ and $[R_\Box]C$ are interpreted by using the corresponding normal operations (i.e. $\Diamond$ and $\Box$, respectively) on the complex algebras of enriched formal contexts. We do not include the concept names $\top$ and $\bot$ in the language, as the naive tableaux rules corresponding to these concepts can lead to non-terminating tableaux algorithms. In the future, we intend to extend the results in this paper to the description logic with concept names $\top$ and $\bot$ in the language. We do not use the symbols $\forall r$ and $\exists r$ in the context of LE-$\mathscr{A}\mathscr{L}\mathscr{C}$, because the semantic clauses of the modal operators in LE-logic use universal quantifiers, and hence using the same notation verbatim would be ambiguous or misleading.

The *TBox axioms* in LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ are of the shape $C_1 \sqsubseteq C_2$, where C_1 and C_2 are concepts[3]. We use $C_1 \equiv C_2$ as a shorthand for $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. The *ABox terms* in LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ are of the form:

$$aR_\Box x, \quad xR_\Diamond a, \quad aIx, \quad a : C, \quad x :: C$$

The *ABox assertions* in LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ are of the form t, $\neg t$, where t is any ABox term. We refer to the terms of first three types and their negations as *relational terms*. We denote an arbitrary LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ *ABox*

[3] As is standard in DL (cf. [2] for more details), general concept inclusion of the form $C_1 \sqsubseteq C_2$ can be rewritten as concept definition $C_1 \equiv C_2 \wedge C_3$, where C_3 is a new concept name.

(resp. *TBox*) with $\mathscr{A}$ (resp $\mathscr{T}$). The interpretations of the ABox assertions $a:C$ and $x::C$ are "object a is a member of concept C" and "feature x is in the description of concept C", respectively. Note that we add the negative terms to ABoxes explicitly, as the LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ concepts do not contain negation. An *interpretation* for the language of LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ is a tuple $\mathscr{M}=(\mathfrak{F},\cdot^{\mathscr{M}})$, where $\mathfrak{F}=(A,X,I^{\mathscr{M}},\{R_{\Box}^{\mathscr{M}} \mid R_{\Box}\in \mathscr{R}_{\Box}\},\{R_{\Diamond}^{\mathscr{M}} \mid R_{\Diamond}\in\mathscr{R}_{\Diamond}\})$ is an enriched formal context and $\cdot^{\mathscr{M}}$ maps:
1. each individual name $a\in \mathsf{OBJ}$ (resp. $x\in \mathsf{FEAT}$), to some $a^{\mathscr{M}}\in A$ (resp. $x^{\mathscr{M}}\in X$) in $\mathfrak{F}$;
2. role names I, $R_{\Box}\in\mathscr{R}_{\Box}$ and $R_{\Diamond}\in\mathscr{R}_{\Diamond}$ to relations $I^{\mathscr{M}}$, $R_{\Box}^{\mathscr{M}}$ and $R_{\Diamond}^{\mathscr{M}}$ in $\mathfrak{F}$, notice that $R_{\Box}^{\mathscr{M}}$ and $R_{\Diamond}^{\mathscr{M}}$ in $\mathfrak{F}$ are all $I^{\mathscr{M}}$-compatible;
3. each primitive concept D to some $D^{\mathscr{M}}\in\mathfrak{F}^{+}$, and other concepts as follows:

$$(C_1\wedge C_2)^{\mathscr{M}}=C_1^{\mathscr{M}}\wedge C_2^{\mathscr{M}}\quad (C_1\vee C_2)^{\mathscr{M}}=C_1^{\mathscr{M}}\vee C_2^{\mathscr{M}}\quad ([R_{\Box}]C)^{\mathscr{M}}=[R_{\Box}^{\mathscr{M}}]C^{\mathscr{M}}\quad (\langle R_{\Diamond}\rangle C)^{\mathscr{M}}=\langle R_{\Diamond}^{\mathscr{M}}\rangle C^{\mathscr{M}}$$

where all the operators are defined as on the complex algebra $\mathfrak{F}^{+}$. The *satisfiability relation* for an interpretation $\mathscr{M}=(\mathfrak{F},\cdot^{\mathscr{M}})$ is defined as follows:
1. $\mathscr{M}\vDash C_1\equiv C_2$ iff $[\![C_1^{\mathscr{M}}]\!]=[\![C_2^{\mathscr{M}}]\!]$ iff $([C_2^{\mathscr{M}}])=([C_1^{\mathscr{M}}])$.
2. $\mathscr{M}\vDash a:C$ iff $a^{\mathscr{M}}\in[\![C^{\mathscr{M}}]\!]$, and $\mathscr{M}\vDash x::C$ iff $x^{\mathscr{M}}\in([C^{\mathscr{M}}])$.
3. $\mathscr{M}\vDash aIx$ (resp. $aR_{\Box}x$, $xR_{\Diamond}a$) iff $a^{\mathscr{M}}I^{\mathscr{M}}x^{\mathscr{M}}$ (resp. $a^{\mathscr{M}}R_{\Box}^{\mathscr{M}}x^{\mathscr{M}}$, $x^{\mathscr{M}}R_{\Diamond}^{\mathscr{M}}a^{\mathscr{M}}$).
4. $\mathscr{M}\vDash\neg\alpha$, where α is any ABox term, iff $\mathscr{M}\nvDash\alpha$.

An LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox (resp. TBox) is a finite set of ABox assertions (resp. TBox axioms) in LE-$\mathscr{A}\mathscr{L}\mathscr{C}$. A *knowledge base* in LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ is a tuple $(\mathscr{A},\mathscr{T})$, where $\mathscr{A}$ is an LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox, and $\mathscr{T}$ is an LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ TBox. An interpretation $\mathscr{M}$ is a *model* for a knowledge base $(\mathscr{A},\mathscr{T})$ if $\mathscr{M}\vDash\mathscr{A}$ and $\mathscr{M}\vDash\mathscr{T}$. A knowledege base $(\mathscr{A},\mathscr{T})$ is *consistent* if there is a model for it. An ABox $\mathscr{A}$ (resp. TBox $\mathscr{T}$) is consistent if there exists a model for knowledege base $(\mathscr{A},\emptyset)$ (resp. $(\emptyset,\mathscr{T})$).

2.3 Tableaux algorithm for checking LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox consistency

In this section, we recall the tableaux algorithm for checking LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox consistency, which is introduced in [7, 8]. We noticed a mistake in the proof of termination and I-compatibility in an earlier version of this paper [7] in which concepts $\top$ and $\bot$ were included as concept names in the language. In the updated version [8], we prove that the result is valid in the restricted language that does not contain $\top$ and $\bot$ in the language. In this paper, we work with the restricted language as in [8].

An LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ ABox $\mathscr{A}$ contains a *clash* if it contains both ABox assertions β and $\neg\beta$. The expansion rules are designed so that the expansion of $\mathscr{A}$ will contain a clash iff $\mathscr{A}$ is inconsistent. The set $sub(C)$ of the sub-formulas of any LE-$\mathscr{A}\mathscr{L}\mathscr{C}$ concept C is defined as usual. A concept C' *occurs* in $\mathscr{A}$, write as $C'\in\mathscr{A}$, if $C'\in sub(C)$ for some C such that one of the ABox assertions $a:C$, $x::C$, $\neg a:C$, or $\neg x::C$ is in $\mathscr{A}$. An individial name b (resp. y) *occurs* in $\mathscr{A}$, write as $b\in\mathscr{A}$ (resp. $y\in\mathscr{A}$), if there is an ABox assertion in $\mathscr{A}$ which contains b (resp. y).

The Algorithm 1 (constructively) provides a model $\mathscr{M}=(\mathfrak{F},\cdot^{\mathscr{M}})$ for every consistent ABox $\mathscr{A}$, where $\mathfrak{F}=(A,X,I,\mathscr{R}_{\Box},\mathscr{R}_{\Diamond})$. This model has the following property:

For any $C\in\mathscr{A}$, there exist $a_C\in A$ and $x_C\in X$ such that, for any $a\in A$ (resp. $x\in X$), $a\in[\![C^{\mathscr{M}}]\!]$ (resp. $x\in([C])^{\mathscr{M}}$) iff aIx_C (resp. a_CIx).

We call a_C and x_C the *classifying object* and the *classifying feature* of C, respectively. To make our notation more readable, we will write $a_{\Box C}$, $x_{\Box C}$ (resp. $a_{\Diamond C}$, $x_{\Diamond C}$) instead of $a_{[R_{\Box}]C}$, $x_{[R_{\Box}]C}$ (resp. $a_{\langle R_{\Diamond}\rangle C}$, $x_{\langle R_{\Diamond}\rangle C}$).

Creation rule

$$\frac{\text{For any } C\in\mathscr{A}}{a_C:C,\quad x_C::C}\ \text{create}$$

Basic rule

$$I\ \frac{b:C,\quad y::C}{bIy}$$

Algorithm 1 tableaux algorithm for checking LE-$\mathscr{ALC}$ ABox consistency

Input: An LE-$\mathscr{ALC}$ ABox $\mathscr{A}$. **Output**: whether $\mathscr{A}$ is inconsistent.

1: **if** there is a clash in $\mathscr{A}$ **then return** "inconsistent".
2: **pick** any applicable expansion rule R, **apply** R to $\mathscr{A}$ and proceed recursively.
3: **if** no expansion rule is applicable **return** "consistent".

Rules for the logical connectives I-compatibility rules

$$\frac{b : C_1 \wedge C_2}{b : C_1, \quad b : C_2}\ \wedge_A \qquad \vee_X \frac{y :: C_1 \vee C_2}{y :: C_1, \quad y :: C_2} \qquad \Box y \frac{bI\Box y}{bR_\Box y} \qquad \frac{bI\blacksquare y}{yR_\Diamond b}\ \blacksquare y$$

$$\Box \frac{b : [R_\Box]C, \quad y :: C}{bR_\Box y} \qquad \frac{y :: \langle R_\Diamond\rangle C, \quad b : C}{yR_\Diamond b}\ \Diamond \qquad \Diamond b \frac{\Diamond bIy}{yR_\Diamond b} \qquad \frac{\blacklozenge bIy}{bR_\Box y}\ \blacklozenge b$$

invese rule for connectives

$$\wedge_A^{-1} \frac{b : C_1, b : C_2, C_1 \wedge C_2 \in \mathscr{A}}{b : C_1 \wedge C_2} \qquad \frac{y :: C_1, y :: C_2, C_1 \vee C_2 \in \mathscr{A}}{y :: C_1 \vee C_2}\ \vee_X^{-1}$$

Adjunction rules

$$R_\Box \frac{bR_\Box y}{\blacklozenge bIy, \quad bI\Box y} \qquad \frac{yR_\Diamond b}{\Diamond bIy, \quad bI\blacksquare y}\ R_\Diamond$$

Basic rules for negative assertions **Appending rules**

$$\neg b \frac{\neg(b : C)}{\neg(bIx_C)} \qquad \frac{\neg(x :: C)}{\neg(a_C Ix)}\ \neg x \qquad x_C \frac{bIx_C}{b : C} \qquad \frac{a_C Iy}{y :: C}\ a_C$$

In the adjunction rules the individuals $\blacklozenge b$, $\Diamond b$, $\Box y$, and $\blacksquare y$ are new and unique for each relation $R_\Box$ and $R_\Diamond$, except for $\Diamond a_C = a_{\Diamond C}$ and $\Box x_C = x_{\Box C}$[4].

The following theorem follows from the results in [8]:

Theorem 1. *Algorithm 1 provides a sound and complete polynomial-time decision procedure for checking consistency of* LE-$\mathscr{ALC}$ *ABoxes.*

Remark 1. *The Algorithm 1 can be extended to an exponential-time algorithm for checking consistency of knowledge bases with acyclic TBoxes via the unraveling technique (cf. [2] for details).*

3 Description logic LE-$\mathscr{FALC}$

In this section, we introduce the fuzzy non-distributive description logic LE-$\mathscr{FALC}$ based on many-valued LE-logic, which can be seen as a fuzzy generalization of description logic LE-$\mathscr{ALC}$.

The *individual names*, *role names*, *primitive concepts*, *concepts* and *ABox terms* in the language of LE-$\mathscr{FALC}$ are the same as those in the language of LE-$\mathscr{ALC}$ (recall Section 2.2), except that they are intended to be interpreted on the complex algebras of **H**-valued enriched formal contexts as described in Section 2.1. The *ABox assertions* in LE-$\mathscr{FALC}$ are of the form:

$$\alpha \leq t, \quad \alpha \not\leq t,$$

where $\alpha \in \mathbf{H}$, and t is an ABox term. For any of the ABox terms t, $\alpha \leq t$ (resp. $\alpha \not\leq t$) is interpreted as t is true at least to the extent α (resp. it is not the case that t is true at least to the extent α). The *TBox axioms in* LE-$\mathscr{FALC}$ are of the form:

$$C_1 \equiv C_2,$$

[4]The new individual names $\blacklozenge b$, $\Diamond b$, $\Box y$, and $\blacksquare y$ appearing in tableaux expansion are purely syntactic entities. Intuitively, they correspond to the classifying objects (resp. features) of the concepts $\blacklozenge\mathbf{b}$, $\Diamond\mathbf{b}$ (resp. $\Box\mathbf{y}$, resp. $\blacksquare\mathbf{y}$), where $\mathbf{b} = (b^{\uparrow\downarrow}, b^{\uparrow})$ (resp. $\mathbf{y} = (y^{\downarrow}, y^{\downarrow\uparrow})$) is the concept generated by b (resp. y), and the operation $\blacklozenge$ (resp.$\blacksquare$) is the left (resp. right) adjoint of operation $\Box$ (resp. $\Diamond$).

where C_1 and C_2 are LE-$\mathscr{FALC}$ concepts.

An *interpretation* for LE-$\mathscr{FALC}$ knowledge base is a tuple $\mathscr{M} = (\mathfrak{F}, \cdot^{\mathscr{M}})$, where $\mathfrak{F} = (A, X, I^{\mathscr{M}}, \{R_\Box^{\mathscr{M}} \mid R_\Box \in \mathscr{R}_\Box\}, \{R_\Diamond^{\mathscr{M}} \mid R_\Diamond \in \mathscr{R}_\Diamond\})$ is an **H**-valued enriched formal context, and $\cdot^{\mathscr{M}}$ maps:
1. any individual name $a \in \mathsf{OBJ}$ (resp. $x \in \mathsf{FEAT}$) to some $a^{\mathscr{M}} \in A$ (resp. $x^{\mathscr{M}} \in X$);
2. role names I, $R_\Box \in \mathscr{R}_\Box$ and $R_\Diamond \in \mathscr{R}_\Diamond$ to **H**-relations $I^{\mathscr{M}}$, $R_\Box^{\mathscr{M}}$ and $R_\Diamond^{\mathscr{M}}$ in $\mathfrak{F}$, notice that $R_\Box^{\mathscr{M}}$ and $R_\Diamond^{\mathscr{M}}$ in $\mathfrak{F}$ are all $I^{\mathscr{M}}$-compatible;
3. any primitive concept D to $D^{\mathscr{M}} \in \mathfrak{F}^+$, and the other concepts as follows:

$$(C_1 \wedge C_2)^{\mathscr{M}} = C_1^{\mathscr{M}} \wedge C_2^{\mathscr{M}} \quad (C_1 \vee C_2)^{\mathscr{M}} = C_1^{\mathscr{M}} \vee C_2^{\mathscr{M}} \quad ([R_\Box]C)^{\mathscr{M}} = [R_\Box^{\mathscr{M}}]C^{\mathscr{M}} \quad (\langle R_\Diamond \rangle C)^{\mathscr{M}} = \langle R_\Diamond^{\mathscr{M}} \rangle C^{\mathscr{M}},$$

where all the connectives are interpreted like in the many-valued LE-logic. For any interpretation $\mathscr{M} = (\mathfrak{F}, \cdot^{\mathscr{M}})$, we define an **H**-valuation $v^{\mathscr{M}}$ for ABox terms as follows:
1. $v^{\mathscr{M}}(a : C) = [\![C]\!](a)$ and $v^{\mathscr{M}}(x :: C) = ([C])(x)$.
2. $v^{\mathscr{M}}(I(a,x)) = I^{\mathscr{M}}(a^{\mathscr{M}}, x^{\mathscr{M}})$, $v^{\mathscr{M}}(R_\Box(a,x)) = R_\Box^{\mathscr{M}}(a^{\mathscr{M}}, x^{\mathscr{M}})$, and $v^{\mathscr{M}}(R_\Diamond(x,a)) = R_\Diamond^{\mathscr{M}}(x^{\mathscr{M}}, a^{\mathscr{M}})$.

Let $\mathscr{M}$ be an interpretation, the *satisfiability relation* for $\mathscr{M}$ is defined as follows:
1. $\mathscr{M} \vDash C_1 \equiv C_2$ iff $C_1^{\mathscr{M}} = C_2^{\mathscr{M}}$.
2. For any ABox term t, $\mathscr{M} \vDash \alpha \leq t$ iff $\alpha \leq v^{\mathscr{M}}(t)$ and $\mathscr{M} \vDash \alpha \not\leq t$ iff $\mathscr{M} \nvDash \alpha \leq t$.

A *knowledge base* in LE-$\mathscr{FALC}$ is a tuple $(\mathscr{A}, \mathscr{T})$, where $\mathscr{A}$ is an LE-$\mathscr{ALC}$ ABox, and $\mathscr{T}$ is an LE-$\mathscr{ALC}$ TBox. An interpretation $\mathscr{M}$ is a *model* for a knowledge base $(\mathscr{A}, \mathscr{T})$ if $\mathscr{M} \vDash \mathscr{A}$ and $\mathscr{M} \vDash \mathscr{T}$. A knowledge base $(\mathscr{A}, \mathscr{T})$ is *consistent* if there is a model for it. An ABox $\mathscr{A}$ (resp. TBox $\mathscr{T}$) is consistent if there exists a model for knowledge base $(\mathscr{A}, \emptyset)$ (resp. $(\emptyset, \mathscr{T})$).

4 Examples of LE-$\mathscr{FALC}$ knowledge bases

We now give two toy examples of knowledge bases regarding categorization of scientific papers into different categories based on keywords appearing in them along with epistemic understanding of resulting categories according to an agent in the language LE-$\mathscr{FALC}$.
The following table lists concepts appearing in the knowledge base.

concept name	symbol	concept name	symbol
Chemistry papers	C_1	Physics papers	C_2
Biology papers	C_3	Natural science papers	C_4
Biochemistry papers	C_5	Multi-disciplinary physics papers	C_6

This knowledge base is based on the following set of features:

feature	symbol	feature	symbol
keywords from chemistry	y_1	keywords from physics	y_2
keywords from biology	y_3	keywords from biochemistry	y_4

Let **H** be the Heyting algebra defined by a totally ordered set $\{0, 1/2, 1\}$. As **H** is linear, we use $t < \alpha$ to denote the ABox assertion $\alpha \not\leq t$ for any ABox term t and $\alpha \in \mathbf{H}$. For any paper P and a feature y, $I(P,y) = 0$ (resp. $I(P,y) = 1/2$, resp. $I(P,y) = 1$) if it contains very few or no (resp. some, resp. many) keywords from the field corresponding to feature y. For example, $I(P,y_1) = 1/2$ if paper P contains some (but not many) keywords from chemistry. Let $R_\Box$ be a relation corresponding to an epistemic agent i defined as follows: For any paper P, and a feature y, $R_\Box(P,y) = 0$ (resp. $R_\Box(P,y) = 1/2$, resp. $R_\Box(P,y) = 1$) if it contains very few or no (resp. some, resp. many) keywords from the field corresponding to feature y according to agent i. The I-compatibility of relation $R_\Box$ is justified as explained in Section 2.1. For any two categories C_1 and C_2, the categories $C_1 \vee C_2$ and $C_1 \wedge C_2$ denote their least common super-category, and greatest common sub-category, respectively. The category $[R_\Box]C_1$, denotes the formal concept generated by the (fuzzy) set of objects that agent i believes to have all features to the extent specified by

intension of C_1 i.e. $[R_\Box]C_1$ can be seen as a concept whose extension is perception of extension of C_1 (given the intension of C_1) according to agent i.

Let $\mathscr{K} = (\mathscr{A}, \mathscr{T})$ be a knowledge base such that $\mathscr{T} = \{C_4 \equiv C_1 \vee C_3, C_2 \equiv C_4 \wedge C_7, C_6 \equiv (C_1 \wedge C_2) \vee (C_2 \wedge C_3)\}$, and $\mathscr{A} = \{1 \leq P_1 : C_2, P_1 : C_6 < 1/2, 1 \leq y_1 :: [R_\Box]C_1, 1 \leq y_3 :: [R_\Box]C_3, P_2 R_\Box y_3 < 1, 1/2 \leq P_2 : C_1, 1/2 \leq y_1 :: C_1\}$, where C_7 is a new fresh concept name. TBox axiom $C_4 \equiv C_1 \vee C_3$ means that the category of natural science papers is the smallest category (in the categorization system defined by given knowledge base) of papers containing both chemistry and biology papers. TBox axiom $C_2 \equiv C_4 \wedge C_7$ states that the physics papers are natural science papers. TBox axiom $C_6 \equiv (C_1 \wedge C_2) \vee (C_2 \wedge C_3)$ states that multi-disciplinary physics papers is the smallest category containing categories of papers which are considered both physics and chemistry papers and physics and biology papers.

ABox assertion $1 \leq P_1 : C_2$ (resp. $1/2 \leq P_2 : C_1$) states that paper P_1 (resp. P_2) is a physics (resp. chemistry) paper to the extent 1 (resp. $1/2$). ABox assertion $P_1 : C_1 < 1/2$ (resp. $P_1 : C_6 < 1/2$) state that paper P_1 is a chemistry (resp. multi-disciplinary physics) paper to extent 0. ABox assertion $1 \leq y_1 :: [R_\Box]C_1$, (resp. $1 \leq y_3 :: [R_\Box]C_3$) states that any paper which is Chemistry (resp. biology) paper according to agent i must have many keywords from chemistry (resp. biology). The ABox assertion $P_2 R_\Box y_3 < 1$ states that paper P_2 does not have many physics keywords according to agent i. ABox assertion $1/2 \leq y_1 :: C_1$ states that having at least some keywords from Chemistry is in intension (defining features) of category of Chemistry papers. Note that the assertions $C_2 \equiv C_4 \wedge C_7$, $C_6 \equiv (C_1 \wedge C_2) \vee (C_2 \wedge C_3)$, $1 \leq P_1 : C_2$, and $P_1 : C_6 < 1/2$ together imply that any model of $\mathscr{K}$ must be non-distributive.

For another example, consider the knowledge base $\mathscr{K}' = (\mathscr{A}', \mathscr{T}')$ such that $\mathscr{T}' = \{C_5 \equiv C_1 \wedge C_3\}$ and $\mathscr{A}' = \{1/2 \leq P_3 : [R_\Box]C_1, 1 \leq P_3 : [R_\Box]C_3, P_3 R_\Box y_4 < 1/2, 1/2 \leq y_4 :: C_5\}$. We can provide interpretations for different axioms of $\mathscr{K}'$ similar to the interpretations of axioms in $\mathscr{K}$ discussed above.

5 Tableaux algorithm for checking consistency of ABoxes

In this section, we introduce the Algorithm 2 which is modified from the Algorithm 1 to check the consistency of LE-$\mathscr{FALC}$ ABoxes. We say that LE-$\mathscr{FALC}$ ABox $\mathscr{A}$ contains a *clash* if there exist $\alpha_1 \leq t$ and $\alpha_2 \not\leq t$ in $\mathscr{A}$ with $\alpha_2 \leq \alpha_1$ for some ABox term t. The set $sub(C)$ of the sub-formulas of any LE-$\mathscr{FALC}$ concept C is defined as usual. A concept C' *occurs* in $\mathscr{A}$, write as $C' \in \mathscr{A}$, if $C' \in sub(C)$ for some C such that there is an ABox assertion in $\mathscr{A}$ which contains C. A constant b (resp. y) *occurs* in $\mathscr{A}$, in symbols, $b \in \mathscr{A}$ (resp. $y \in \mathscr{A}$), if there is an ABox assertion in $\mathscr{A}$ which contains b (resp. y).

For any concept $C \in \mathscr{A}$, the added constants a_C and x_C act as *classifying object* and *classifying feature* of concept C, in the sense that if $\mathscr{A}$ is consistent, then the tableaux algorithm will construct a model which satisfies $[\![C]\!](b) = \max\{\alpha \mid \alpha \leq I(b, x_C) \in \overline{\mathscr{A}}\}$, and $([C])(y) = \max\{\alpha \mid \alpha \leq I(a_C, y) \in \overline{\mathscr{A}}\}$ for any object b, and feature y in the model. The set of tableaux expansion rules for LE-$\mathscr{FALC}$ is listed below. The commas in each rule are meta-linguistic conjunctions, hence every tableau is non-branching.

Creation rule

$$\frac{\text{For any } C \in \mathscr{A}}{1 \leq a_C : C, \quad 1 \leq x_C :: C} \text{ create}$$

Basic rule

$$I\ \frac{\alpha_1 \leq b : C, \quad \alpha_2 \leq y :: C}{\alpha_1 \wedge \alpha_2 \leq I(b, y)}$$

Rules for the logical connectives

$$\frac{\alpha \leq b : C_1 \wedge C_2}{\alpha \leq b : C_1, \quad \alpha \leq b : C_2} \wedge_A \qquad \vee_X\ \frac{\alpha \leq y :: C_1 \vee C_2}{\alpha \leq y :: C_1, \quad \alpha \leq y :: C_2}$$

$$\Box\ \frac{\alpha_1 \leq b : [R_\Box]C, \quad \alpha_2 \leq y :: C}{\alpha_1 \wedge \alpha_2 \leq R_\Box(b, y)} \qquad \frac{\alpha_1 \leq y :: \langle R_\Diamond \rangle C, \quad \alpha_2 \leq b : C}{\alpha_1 \wedge \alpha_2 \leq R_\Diamond(y, b)}\ \Diamond$$

I-compatibility rules

$$\Box y\ \frac{\alpha \leq I(b,\Box y)}{\alpha \leq R_{\Box}(b,y)} \qquad \frac{\alpha \leq I(b,\blacksquare y)}{\alpha \leq R_{\Diamond}(y,b)}\ \blacksquare y \qquad \Diamond b\ \frac{\alpha \leq I(\Diamond b,y)}{\alpha \leq R_{\Diamond}(y,b)} \qquad \frac{\alpha \leq I(\blacklozenge b,y)}{cR_{\Box}(b,y)}\ \blacklozenge b$$

inverse rule for connectives

$$\wedge_A^{-1}\ \frac{\alpha_1 \leq b : C_1,\ \alpha_2 \leq b : C_2;\ \text{where } C_1 \wedge C_2 \in \mathscr{A}}{\alpha_1 \wedge \alpha_2 \leq b : C_1 \wedge C_2} \qquad \frac{\alpha_1 \leq y :: C_1,\ \alpha_2 \leq y :: C_2;\ \text{where } C_1 \vee C_2 \in \mathscr{A}}{\alpha_1 \wedge \alpha_2 \leq y :: C_1 \vee C_2}\ \vee_X^{-1}$$

Adjunction rules

$$R_{\Box}\ \frac{\alpha \leq R_{\Box}(b,y)}{\alpha \leq I(\blacklozenge b,y),\quad \alpha \leq I(b,\Box y)} \qquad \frac{\alpha \leq R_{\Diamond}(y,b)}{\alpha \leq I(\Diamond b,y),\quad \alpha \leq I(b,\blacksquare y)}\ R_{\Diamond}$$

Basic rules for negative assertions **Appending rules**

$$-a_C\ \frac{\alpha \not\leq (b : C)}{\alpha \not\leq I(b,x_C)} \qquad -x_C\ \frac{\alpha \not\leq (y :: C)}{\alpha \not\leq I(a_C,y)} \qquad x_C\ \frac{\alpha \leq I(b,x_C)}{\alpha \leq b : C} \qquad \frac{\alpha \leq I(a_C,y)}{\alpha \leq y :: C}\ a_C$$

Many-valued algebra rules

$$MV_{\vee}\ \frac{\alpha_1 \leq t,\quad \alpha_2 \leq t}{\alpha_1 \vee \alpha_2 \leq t}$$

In the adjunction rules the individual names $\blacklozenge b$, $\Diamond b$, $\Box y$, and $\blacksquare y$ must be new and unique for each relation $R_{\Box}$ and $R_{\Diamond}$, except for $\Diamond a_C = a_{\Diamond C}$ and $\Box x_C = x_{\Box C}$[5]. Side conditions for rules $\wedge_A^{-1}$, and $\vee_X^{-1}$ ensure we do not add new joins or meets to concept names.

Algorithm 2 tableaux algorithm for checking LE-$\mathscr{FALC}$ ABox consistency

Input: An LE-$\mathscr{FALC}$ ABox $\mathscr{A}$. **Output**: whether $\mathscr{A}$ is inconsistent.

1: **if** there is a clash in $\mathscr{A}$ **then return** "inconsistent".
2: **pick** any applicable expansion rule R, **apply** R to $\mathscr{A}$ and proceed recursively.
3: **if** no expansion rule is applicable **return** "consistent".

From the shape of the expansion rules, it is clear that the new terms are added by LE-$\mathscr{FALC}$ expansion rules in a manner identical to LE-$\mathscr{ALC}$ expansion rules except for the rule $MV_{\vee}$. The rule $MV_{\vee}$ at most adds one term for two terms already present in the tableau. Hence, this rule can only increase the size of tableau linearly. Moreover, application of any of the tableaux rules only involves application of a fixed (finite) number of **H** operations. Hence, the polynomial time termination result for LE-$\mathscr{FALC}$ tableaux algorithm follows from the termination result for LE-$\mathscr{ALC}$ tableaux algorithm (See [8, Section 4.1], for more details).

Theorem 2 (Termination)**.** *For any ABox $\mathscr{A}$, the tableaux algorithm 2 terminates in a finite number of steps which is polynomial in size($\mathscr{A}$).*

Similarly to LE-$\mathscr{ALC}$ (see Remark 1), Algorithm 2 can be extended to acyclic TBoxes (exponential-time) via unraveling. In the following sections, we prove the completeness and soundness for the Algorithm 2.

6 Soundness of the tableaux algorithm

For any consistent ABox $\mathscr{A}$, we let its *completion* $\overline{\mathscr{A}}$ be its maximal expansion (which exists due to termination). If there is no clash in $\overline{\mathscr{A}}$, we construct a tuple $\mathscr{M} = (\mathfrak{F}, \cdot^{\mathscr{M}})$ as follows:

[5]The new individual names $\blacklozenge b$, $\Diamond b$, $\Box y$, and $\blacksquare y$ appearing in tableaux expansion are purely syntactic entities. Intuitively, they correspond to the classifying objects (resp. features) of the concepts $\blacklozenge\mathbf{b}$, $\Diamond\mathbf{b}$ (resp. $\Box\mathbf{y}$, resp. $\blacksquare\mathbf{y}$), where $\mathbf{b} = (b^{\uparrow\downarrow}, b^{\uparrow})$ (resp. $\mathbf{y} = (y^{\downarrow}, y^{\downarrow\uparrow})$) is the concept generated by b (resp. y), and the operation $\blacklozenge$ (resp.$\blacksquare$) is the left (resp. right) adjoint of operation $\Box$ (resp. $\Diamond$).

1. A and X are the sets of individual names of objects and features occurring in $\overline{\mathscr{A}}$, respectively.
2. For any $a \in A$, $x \in X$, and any role names $R_\Box \in \mathscr{R}_\Box$, $R_\Diamond \in \mathscr{R}_\Diamond$ we have the values of the maps $I(a,x)$, $R_\Box(a,x)$, $R_\Diamond(a,x)$ are the maximum α_i such that ABox assertions $\alpha_i \leq I(a,x)$, $\alpha_i \leq R_\Box(a,x)$, $\alpha_i \leq R_\Diamond(a,x)$ explicitly occur in $\overline{\mathscr{A}}$.
3. Let $\mathfrak{F} = (A,X,I,\mathscr{R}_\Box,\mathscr{R}_\Diamond)$ be the tuple obtained in this manner.
4. We add a new element $x_\perp$ (resp. $a_\top$) to X (resp. A) such that it is related to any element of A (resp. X) to extent 0 w.r.t. all the relations.
5. The map $\cdot^{\mathscr{M}}$ is defined as follows: For any individual name a (resp. x), we let $a^{\mathscr{M}} := a$ (resp. $x^{\mathscr{M}} := x$). For any primitive concept $D \in \mathscr{A}$, we define $D^{\mathscr{M}} = (\{1/x_D\}^\downarrow, \{1/a_D\}^\uparrow)$.

Next, we show that $\mathscr{M}$ is a model for the ABox $\mathscr{A}$. To this end, we need to show that $\mathfrak{F}$ is an $\mathbf{H}$-valued enriched formal context, i.e. that $R_\Box$ and $R_\Diamond$ are I-compatible, and $D^{\mathscr{M}}$ is a fuzzy formal concept in the complex algebra $\mathfrak{F}^+$ for every atomic concept D. The latter is shown in the next lemma, and the former in the subsequent one.

Lemma 1. *$\{1/x_D\}^{\downarrow\uparrow} = \{1/a_D\}^\uparrow$ and $\{1/a_D\}^{\uparrow\downarrow} = \{1/x_D\}^\downarrow$ for any primitive concept $D \in \mathscr{A}$.*

Proof. We only prove the first equation here, and the second equation can be proved similarly. It sucffices to prove that for any $y \in X$, $\{1/x_D\}^{\downarrow\uparrow}(y) = \{1/a_D\}^\uparrow(y)$. Note that for any $y \in X$, and $b \in A$,

$$\{1/x_D\}^{\downarrow\uparrow}(y) = \wedge_{b\in A}(I(b,x_D) \to I(b,y)), \tag{3}$$

$$\{1/a_D\}^{\downarrow\uparrow}(b) = \wedge_{y\in X}(I(a_D,y) \to I(b,y)). \tag{4}$$

By the creation rules, we always have $1 \leq a_D : D$ and $1 \leq x_D :: D$ in $\overline{\mathscr{A}}$. Then by rule I, we have that $1 \leq I(a_D,x_D)$ in $\overline{\mathscr{A}}$. Then by construction and properties of $\mathbf{H}$, for any $y \in X$, $I(a_D,x_D) \to I(a_D,y) = I(a_D,y)$. Therefore, by Equation (3), we have $\{1/x_D\}^{\downarrow\uparrow}(y) \leq I(a_D,y) = \{1/a_D\}^\uparrow(y)$.

For the reverse direction, suppose $\alpha_1 \leq \{1/a_D\}(y) = I(a_D,y)$. We need to show that $\alpha_1 \leq \{1/x_D\}^{\downarrow\uparrow}(y)$ for every y occurring in $\overline{\mathscr{A}}$. By Equation (3), it is enough to show that for every $b \in A$, $\alpha_1 \leq I(b,x_D) \to I(b,y)$. By adjunction, this is equivalent to showing $\alpha_1 \wedge I(b,x_D) \leq I(b,y)$ for every $b \in A$. If term bIx_D does not occur in $\overline{\mathscr{A}}$, by construction, we have $I(b,x_D) = 0$ which trivially implies the required condition. Suppose $I(b,x_D) = \alpha_2 \neq 0$, then by construction $\alpha_2 \leq I(b,x_D) \in \overline{\mathscr{A}}$. As $\alpha_1 \leq I(a_D,y)$, by construction $\alpha_3 \leq I(a_D,y) \in \overline{\mathscr{A}}$ for some $\alpha_1 \leq \alpha_3$. Therefore, by rule I, we have $\alpha_2 \wedge \alpha_3 \leq I(b,y) \in \overline{\mathscr{A}}$. Then by construction, we have $\alpha_1 \wedge \alpha_2 \leq \alpha_2 \wedge \alpha_3 \leq I(b,y)$. Hence proved. □

Lemma 2. *The relations $R_\Box \in \mathscr{R}_\Box$ and $R_\Diamond \in \mathscr{R}_\Diamond$ in $\mathfrak{F} = (\mathfrak{P},\mathscr{R}_\Box,\mathscr{R}_\Diamond)$ are I-compatible.*

Proof. We prove that $R_\Box^{(0)}[\{\alpha/y\}]$ is Galois-stable for every y occurring in $\overline{\mathscr{A}}$, and any $\alpha \in \mathbf{H}$. The proofs for other conditions are similar. We consider two cases: whether $\Box y \in \overline{\mathscr{A}}$ or not.

When $\Box y \in \overline{\mathscr{A}}$, for any $\beta \in \mathbf{H}$ and $b \in A$, if $\beta \leq I(b,\Box y)$ (resp. $\beta \leq R_\Box(b,y)$) occurs in $\overline{\mathscr{A}}$, then $\beta \leq R_\Box(b,y)$ (resp. $\beta \leq I(b,\Box y)$) occurs in $\overline{\mathscr{A}}$ by $\Box y$ (resp. $R_\Box$) rule. Therefore, by construction $I(b,\Box y) = R_\Box(b,y)$ for any $b \in A$, which implies $R_\Box^{(0)}[\{\alpha/y\}](b) = I^{(0)}[\{\alpha/\Box y\}](b)$ for any $b \in A$ and $\alpha \in \mathbf{H}$. Therefore, for any $\alpha \in \mathbf{H}$, $R_\Box^{(0)}[\{\alpha/y\}] = I^{(0)}[\{\alpha/\Box y\}]$, and hence $R_\Box^{(0)}[\{\alpha/y\}]$ is Galois-stable. When $\Box y$ does not occur in $\overline{\mathscr{A}}$, no term of form $R_\Box(b,y)$ occurs in $\overline{\mathscr{A}}$. In such case, we have for any α, $R_\Box^{(0)}[\{\alpha/y\}] = I^{(0)}[\{1/x_\perp\}] = \emptyset$ is Galois-stable. □

From the above lemmas, it immediately follows that the tuple $\mathscr{M} = (\mathfrak{F},\cdot^{\mathscr{M}})$ defined at the beginning of the present section is an interpretation for LE-$\mathscr{FALC}$. The following lemma states that the interpretation of any concept C in $\mathscr{M}$ is completely determined by the ABox terms of the form $\alpha \leq I(b,x_C)$ and $\alpha \leq I(a_C,y)$ occurring in $\overline{\mathscr{A}}$.

Lemma 3. *Let $\mathscr{M} = (\mathfrak{F}, \cdot^{\mathscr{M}})$ be the interpretation defined by the construction above. Then for any concept C and individuals b, y that occur in $\mathscr{A}$, we have $[\![C]\!](b) = \max\{\alpha \mid \alpha \leq I(b, x_C) \in \overline{\mathscr{A}}\}$, and $([C])(y) = \max\{\alpha \mid \alpha \leq I(a_C, y) \in \overline{\mathscr{A}}\}$.*

Proof. See Appendix A.1. □

Theorem 3 (Soundness)**.** *Let $\mathscr{A}$ be an* LE-$\mathscr{FALC}$ *ABox and $\overline{\mathscr{A}}$ be its completion which does not contain any clash, then the interpretation $\mathscr{M} = (\mathfrak{F}, \cdot^{\mathscr{M}})$ defined above is a model for $\mathscr{A}$.*

Proof. We show that $\mathscr{M}$ is a model for $\overline{\mathscr{A}}$, so it is a model for $\mathscr{A}$. The proof is by cases.
1. By construction, $\mathscr{M}$ satisfies all the ABox assertions of the form $\alpha \leq t$ in $\overline{\mathscr{A}}$, where t is an ABox term of the form $I(a,x)$, $R_{\Box}(a,x)$ or $R_{\Diamond}(a,x)$.
2. Let $\alpha \not\leq t$ be an ABox assertion in $\overline{\mathscr{A}}$, where t is a relational term. By construction, $\mathscr{M} \nvDash \alpha \not\leq t$ iff $\alpha' \leq t \in \overline{\mathscr{A}}$ for some $\alpha \leq \alpha'$. However, if such ABox assertion $\alpha' \leq t$ occurs in $\overline{\mathscr{A}}$, it clashes with the ABox assertion $\alpha \not\leq t$. Therefore, as $\overline{\mathscr{A}}$ contains no clash, we have $\mathscr{M} \vDash \alpha \not\leq t$.
3. For the ABox assertion of the form $\alpha \leq b : C$ (resp. $\alpha \leq y :: C$) occurring in $\overline{\mathscr{A}}$, by expansion rules the ABox assertion of the form $\alpha \leq I(b, x_C)$ (resp. $\alpha \leq I(a_C, y)$) occurs in $\overline{\mathscr{A}}$. By Lemma 3, in the interpretation $\mathscr{M}$, $\alpha \leq [\![C]\!](b)$ (resp. $\alpha \leq ([C])(y)$), which means that $\mathscr{M} \vDash \alpha \leq b : C$ (resp. $\mathscr{M} \vDash \alpha \leq y :: C$).
4. For ABox assertions of the form $\alpha \not\leq b : C$ (resp. $\alpha \not\leq y :: C$) occurring in $\overline{\mathscr{A}}$, by Lemma 3, $\mathscr{M} \nvDash \alpha \not\leq b : C$ (resp. $\mathscr{M} \nvDash \alpha \not\leq y :: C$) iff some term of the form $\alpha' \leq b : C$ (resp. $\alpha' \leq y :: C$) occurs in $\overline{\mathscr{A}}$ for some $\alpha \leq \alpha'$, which implies $\overline{\mathscr{A}}$ contains a clash. Therefore, $\mathscr{M} \vDash \alpha \not\leq b : C$ (resp. $\mathscr{M} \vDash \alpha \not\leq y :: C$) because $\overline{\mathscr{A}}$ contains no clash. □

The following corollary is an immediate consequence of the termination and soundness of the tableaux algorithm.

Corollary 1 (Finite Model Property)**.** *For any consistent* LE-$\mathscr{FALC}$ *ABox $\mathscr{A}$, there exists a model of $\mathscr{A}$ of size polynomial in size($\mathscr{A}$).*

Proof. The interpretation $\mathscr{M}$ defined above is the required model. The polynomial bound on the size of $\mathscr{M}$ follows from the proof of Theorem 2. □

7 Completeness of the tableaux algorithm

In this section, we prove the completeness of the tableaux algorithm given in Section 5. The following lemma is key to this end, since it shows that every model for an LE-$\mathscr{FALC}$ ABox can be extended to a model with classifying objects and features.

Lemma 4. *For any ABox $\mathscr{A}$, any model $\mathscr{M} = (\mathfrak{F}, \cdot^{\mathscr{M}})$ of $\mathscr{A}$ can be extended to a model $\mathscr{M}' = (\mathfrak{F}', \cdot^{\mathscr{M}'})$ of $\mathscr{A}$ such that $\mathfrak{F}' = (A', X', I', \{R'_{\Box}\}_{\Box \in \mathscr{G}}, \{R'_{\Diamond}\}_{\Diamond \in \mathscr{F}})$, $A \subseteq A'$ and $X \subseteq X'$, and moreover for every $\Box \in \mathscr{G}$ and $\Diamond \in \mathscr{F}$:*

1. For any concept C, there exists $a_C \in A'$ and $x_C \in X'$ such that:

$$C^{\mathscr{M}'} = (I'^{(0)}[\{1/x_C\}], I'^{(1)}[\{1/a_C\}]), \quad [\![C^{\mathscr{M}'}]\!](a_C) = 1, \quad ([C^{\mathscr{M}'}])(x_C) = 1, \tag{5}$$

2. For every individual $b \in A$, and $\alpha \in \mathbf{H}$, there exist $\Diamond b, \blacklozenge b \in A'$ such that:

$$I'^{(1)}[\{\alpha/\blacklozenge b\}] = R'^{(1)}_{\Box}[\{\alpha/b^{\mathscr{M}'}\}] \quad \textit{and} \quad I'^{(1)}[\{\alpha/\Diamond b\}] = R'^{(0)}_{\Diamond}[\{\alpha/b^{\mathscr{M}'}\}], \tag{6}$$

3. For every individual $y \in X$, and $\alpha \in \mathbf{H}$, there exist $\Box y, \blacksquare y \in X'$ such that:

$$I'^{(0)}[\{\alpha/\blacksquare y\}] = R'^{(1)}_{\Diamond}[\{\alpha/y^{\mathscr{M}'}\}] \quad \textit{and} \quad I'^{(0)}[\{\alpha/\Box y\}] = R'^{(0)}_{\Box}[\{\alpha/y^{\mathscr{M}'}\}]. \tag{7}$$

4. For any concept C, and any $a \in A$, $x \in X$,

$$[\![C^{\mathscr{M}}]\!](a) = [\![C^{\mathscr{M}'}]\!](a) \quad \textit{and} \quad ([C^{\mathscr{M}}])(x) = ([C^{\mathscr{M}'}])(x). \tag{8}$$

Proof. Fix $\Box \in \mathscr{G}$ and $\Diamond \in \mathscr{F}$. Let $\mathscr{M}' = (\mathfrak{F}', \cdot^{\mathscr{M}'})$, where $\mathfrak{F}' = (A', X', I', \{R'_{\Box}\}_{\Box \in \mathscr{G}}, \{R'_{\Diamond}\}_{\Diamond \in \mathscr{F}})$, be defined as follows: For every concept C, we add new elements a_C and x_C to A and X (respectively) to obtain the sets A' and X'. For any $J \in \{I, R_{\Box}\}$, $a \in A'$ and $x \in X'$, the value of $J'(a,x)$ is defined as follows:
1. If $a \in A$, $x \in X$, then $J'(a,x) = J(a,x)$;
2. If $x \in X$, and $a = a_C$ for some concept C, then $J'(a,x) = \bigwedge_{b \in A}([\![C^{\mathscr{M}}]\!](b) \to J(b,x))$;
3. If $a \in A$, and $x = x_C$ for some concept C, then $J(a,x) = \bigwedge_{y \in X}(([C^{\mathscr{M}}])(y) \to J(a,y))$;
4. If $a = a_{C_1}$ and $x = x_{C_2}$ for some concepts C_1, C_2, then $J'(a,x) = \bigwedge_{b \in A}\bigwedge_{y \in X}((([\![C_1^{\mathscr{M}}]\!](b) \wedge ([C^{\mathscr{M}}])(y)) \to J(b,y))$.

For any $a \in A'$ and $x \in X'$, the value of $R'_{\Diamond}(a,x)$ is defined as follows:
1. If $a \in A$, $x \in X$, then $R'_{\Diamond}(x,a) = R_{\Diamond}(x,a)$;
2. If $x \in X$, and $a = a_C$ for some concept C, then $R_{\Diamond}(x,a) = \bigwedge_{b \in A}([\![C^{\mathscr{M}}]\!](b) \to R_{\Diamond}(x,b))$;
3. If $a \in A$, and $x = x_C$ for some concept C, then $R_{\Diamond}(x,a) = \bigwedge_{y \in X}(([C^{\mathscr{M}}])(y) \to R_{\Diamond}(y,a))$;
4. If $a = a_{C_1}$ and $x = x_{C_2}$ for some concepts C_1, C_2, then $R_{\Diamond}(x,a) = \bigwedge_{b \in A}\bigwedge_{y \in X}((([\![C_1^{\mathscr{M}}]\!](b) \wedge ([C^{\mathscr{M}}])(y)) \to R_{\Diamond}(y,b))$.

For any $b \in A$, $y \in X$, let $\blacklozenge b = a_{\blacklozenge(cl(b))}$, $\Diamond b = a_{\Diamond(cl(b))}$, $\blacksquare y = x_{\blacksquare(cl(y))}$, and $\Box y = x_{\Box(cl(y))}$, where $cl(b)$ (resp. $cl(y)$) is the smallest fuzzy formal concept generated by $\{1/b\}$ (resp. $\{1/y\}$), and the operations $\blacklozenge$ and $\blacksquare$ are the adjoints of the operations $\Box$, and $\Diamond$, respectively. For any concept C, let $C^{\mathscr{M}'} = (I'^{(0)}[\{1/x_C\}], I'^{(1)}[\{1/a_C\}])$.

It is straightforward to check that $\mathscr{M}'$ is a model for $\mathscr{A}$ and satisfies all the properties required in Lemma 4. This concludes the proof. □

Theorem 4 (Completeness)**.** *Let $\mathscr{A}$ be a consistent ABox and $\mathscr{A}'$ be obtained via the application of any expansion rule applied to $\mathscr{A}$, then $\mathscr{A}'$ is also consistent.*

Proof. If $\mathscr{A}$ is consistent, by Lemma 4, there exists a model $\mathscr{M}'$ of $\mathscr{A}$ which satisfies (5), (6) and (7). The theorem follows from the fact that any ABox assertion added by any expansion rule is satisfied by $\mathscr{M}'$, where we interpret a_C, x_C, $\blacklozenge b$, $\Diamond b$, $\Box y$, $\blacksquare y$ as in Lemma 4. □

As a demonstration of the functioning of the tableaux algorithm, we use tableaux Algorithm 2 (using unraveling to deal with TBox axioms) to show that the knowledge base $\mathscr{K}$ discussed in Section 4 is consistent and construct a model for it (cf. Appendix B), while the knowledge base $\mathscr{K}'$ discussed in Section 4 is inconsistent (cf. Appendix C for the proof).

8 Conclusions and future directions

In this paper, we define a fuzzy lattice-based non-distributive description logic LE-$\mathscr{FALC}$ as a description logic to describe and reason about fuzzy formal concepts arising from fuzzy (enriched) formal contexts, and define a polynomial-time tableaux algorithm to check the consistency of LE-$\mathscr{FALC}$

ABoxes. Additionally, this algorithm can be extended to an exponential-time algorithm for checking consistency of knowledge bases with acyclic TBoxes. This work can be extended in several interesting directions, including, but not limited to, the following.

Dealing with cyclic TBoxes and RBox axioms. In this paper, we introduced a tableaux algorithm only for knowledge bases with acyclic TBoxes. In the future, we intend to generalize the algorithm to deal with cyclic TBoxes as well. Another interesting avenue of research is to develop tableaux algorithms for extensions of LE-$\mathscr{FALC}$ with RBox axioms. RBox axioms are used in description logics to describe the relationship between different relations in knowledge bases and the properties of these relations such as reflexivity, symmetry, and transitivity. It would be interesting to see if it is possible to obtain necessary and/or sufficient conditions on the shape of RBox axioms for which a tableaux algorithm can be obtained. This has an interesting relationship with the problem in LE-logic of providing computationally efficient proof systems for various extensions of LE-logic in a modular manner [16, 6].

Generalizing to more expressive description logics. The DL LE-$\mathscr{ALC}$ is the non-distributive counterpart of $\mathscr{ALC}$. A natural direction for further research is to explore the non-distributive counterparts of extensions of $\mathscr{ALC}$ such as $\mathscr{ALCI}$ and $\mathscr{ALCIN}$ and fuzzy generalizations of such description logics. This would allow us to express more constructions like (fuzzy) concepts generated by an object or a feature, which can not be expressed in (LE-$\mathscr{FALC}$) LE-$\mathscr{ALC}$.

Description logic and Formal Concept Analysis. The relationship between FCA and DL has been studied and used in several applications [1, 4, 18]. The framework of LE-$\mathscr{FALC}$ formally brings Fuzzy FCA and DL together, both because its concepts are naturally interpreted as formal concepts in **H**-valued FCA, and because its language is designed to represent knowledge and reasoning in enriched formal contexts. Thus, these results can help integrate FCA and DL further in both theory and applications.

References

[1] Jamal Atif, Celine Hudelot & Isabelle Bloch (2014): *Explanatory Reasoning for Image Understanding Using Formal Concept Analysis and Description Logics*. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 44(5), pp. 552–570, doi:10.1109/TSMC.2013.2280440.

[2] Franz Baader (2003): *The description logic handbook: Theory, implementation and applications*. Cambridge university press.

[3] Franz Baader, Ian Horrocks, Carsten Lutz & Uli Sattler (2017): *Introduction to description logic*. Cambridge University Press, doi:10.1017/9781139025355.

[4] Franz Baader & Baris Sertkaya (2004): *Applying Formal Concept Analysis to Description Logics*. *Concept Lattices*, pp. 261–286, doi:10.1007/978-3-540-24651-0_24.

[5] Radim Bêlohlávek (1999): *Fuzzy Galois Connections*. *Mathematical Logic Quarterly* 45(4), pp. 497–504, doi:10.1002/malq.19990450408.

[6] Ineke van der Berg, Andrea De Domenico, Giuseppe Greco, Krishna B. Manoorkar, Alessandra Palmigiano & Mattia Panettiere (2023): *Labelled Calculi for the Logics of Rough Concepts*. *Logic and Its Applications*, pp. 172–188, doi:10.1007/978-3-031-26689-8_13.

[7] Ineke van der Berg, Andrea De Domenico, Giuseppe Greco, Krishna B Manoorkar, Alessandra Palmigiano & Mattia Panettiere (2023): *Non-distributive description logic*. In: *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, Springer Nature Switzerland Cham, pp. 49–69, doi:10.1007/978-3-031-43513-3_4.

[8] Ineke van der Berg, Andrea De Domenico, Giuseppe Greco, Krishna B. Manoorkar, Alessandra Palmigiano & Mattia Panettiere (2024): *Non-distributive description logic*. arXiv:2307.09561.

[9] Willem Conradie, Sabine Frittella, Krishna Manoorkar, Sajad Nazari, Alessandra Palmigiano, Apostolos Tzimoulis & Nachoem M Wijnberg (2021): *Rough concepts*. *Information Sciences* 561, pp. 371–413, doi:10.1016/j.ins.2020.05.074.

[10] Willem Conradie, Sabine Frittella, Alessandra Palmigiano, Michele Piazzai, Apostolos Tzimoulis & Nachoem M Wijnberg (2016): *Categories: how I learned to stop worrying and love two sorts*. In: *Logic, Language, Information, and Computation: 23rd International Workshop, WoLLIC 2016, Puebla, Mexico, August 16-19th, 2016. Proceedings 23*, Springer, pp. 145–164, doi:10.1007/978-3-662-52921-8_10.

[11] Willem Conradie, Sabine Frittella, Alessandra Palmigiano, Michele Piazzai, Apostolos Tzimoulis & Nachoem M Wijnberg (2017): *Toward an epistemic-logical theory of categorization*. *arXiv preprint arXiv:1707.08743*.

[12] Willem Conradie, Alessandra Palmigiano, Claudette Robinson, Apostolos Tzimoulis & Nachoem M Wijnberg (2019): *The logic of vague categories*. *arXiv preprint arXiv:1908.04816*.

[13] Willem Conradie, Alessandra Palmigiano, Claudette Robinson & Nachoem Wijnberg (2020): *Non-distributive logics: from semantics to meaning*. *arXiv preprint arXiv:2002.04257*.

[14] Bernhard Ganter & Rudolf Wille (1997): *Applied lattice theory: Formal concept analysis*. In: *In General Lattice Theory, G. Grätzer editor, Birkhäuser*, Citeseer.

[15] Bernhard Ganter & Rudolf Wille (2012): *Formal concept analysis: mathematical foundations*. Springer Science & Business Media.

[16] Giuseppe Greco, Minghui Ma, Alessandra Palmigiano, Apostolos Tzimoulis & Zhiguang Zhao (2016): *Unified correspondence as a proof-theoretic tool*. *Journal of Logic and Computation* 28(7), p. 1367–1442.

[17] Petr Hájek (1998): *Metamathematics of Fuzzy Logic*. Kluwer Academic Publishers, Dordrecht, Boston and London, doi:10.1007/978-94-011-5300-3.

[18] Yuncheng Jiang (2019): *Semantifying formal concept analysis using description logics*. *Knowledge-Based Systems* 186, doi:10.1016/j.knosys.2019.104967.

[19] Sergei O. Kuznetsov Jonas Poelmans, Dmitry I. Ignatov & Guido Dedene (2014): *Fuzzy and rough formal concept analysis: a survey*. *International Journal of General Systems* 43(2), pp. 105–134, doi:10.1080/03081079.2013.862377.

[20] Jonas Poelmans, Sergei O Kuznetsov, Dmitry I Ignatov & Guido Dedene (2013): *Formal concept analysis in knowledge processing: A survey on models and techniques*. *Expert systems with applications* 40(16), pp. 6601–6623, doi:10.1016/j.eswa.2013.05.007.

A Proofs

In this appendix, we gather proofs of some results stated throughout the paper.

A.1 Proof of lemma 3

Proof. Note that by rule $MV_\vee$, the maximums in the statement of the lemma exist and are unique when the algorithm terminates. The proof is by induction on the complexity of concept C. The base case (when C is a primitive concept) is immediate by construction of the model. For the induction step, we have four cases.

1. Suppose $C = C_1 \wedge C_2$.

1.1. For the first claim, it is required to prove that

$$\max\{\alpha \mid \alpha \leq I(b, x_{C_1 \wedge C_2}) \in \overline{\mathscr{A}}\} = [\![C_1 \wedge C_2]\!](b).$$

For direction ($\leq$), suppose $\alpha_0 \leq bIx_{C_1\wedge C_2} \in \overline{\mathscr{A}}$ for some α_0. Then by using appending and $\wedge_A$ rules, we have $\alpha_0 \leq I(b,x_{C_1}) \in \overline{\mathscr{A}}$ and $\alpha_0 \leq I(b,x_{C_2}) \in \overline{\mathscr{A}}$. Therefore, $\alpha_0 \leq \max\{\alpha \mid \alpha \leq I(b,x_{C_1})\} \in \overline{\mathscr{A}}\}$, and $\alpha_0 \leq \max\{\alpha \mid \alpha \leq I(b,x_{C_2})\} \in \overline{\mathscr{A}}\}$. Hence, by introduction hypothesis, we have $\alpha_0 \leq [\![C_1]\!](b)$ and $\alpha_0 \leq [\![C_2]\!](b)$, which implies $\alpha_0 \leq [\![C_1]\!](b) \wedge [\![C_2]\!](b) = [\![C_1 \wedge C_2]\!](b)$. Therefore, we have $\max\{\alpha \mid \alpha \leq I(b,x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}\} \leq [\![C_1 \wedge C_2]\!](b)$.

For direction ($\geq$), suppose $\alpha_0 \leq [\![C_1 \wedge C_2]\!](b) = [\![C_1]\!](b) \wedge [\![C_2]\!](b)$. Therefore, we have $\alpha_0 \leq [\![C_1]\!](b)$, and $\alpha_0 \leq [\![C_2]\!](b)$. Hence, by induction hypothesis, there are $\alpha_1, \alpha_2 \in \mathbf{H}$, such that $\alpha_0 \leq \alpha_1$, $\alpha_0 \leq \alpha_2$, $\alpha_1 \leq I(b,x_{C_1})$, and $\alpha_2 \leq I(b,x_{C_2}) \in \overline{\mathscr{A}}$. As $C_1 \wedge C_2$ occurs in $\overline{\mathscr{A}}$, by appending and $\wedge_A^{-1}$ rules, we get $\alpha_1 \wedge \alpha_2 \leq b : C_1 \wedge C_2 \in \overline{\mathscr{A}}$. By creation and the basic rules, this implies $\alpha_1 \wedge \alpha_2 \leq bIx_{C_1\wedge C_2} \in \overline{\mathscr{A}}$. Therefore, we have $\alpha_0 \leq \max\{\alpha \mid \alpha \leq I(b,x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}\}$. Hence, by completeness of $\mathbf{H}$, we have $[\![C_1 \wedge C_2]\!](b) \leq \max\{\alpha \mid \alpha \leq I(b,x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}\}$.

1.2. For the second claim, it is required to prove that

$$([C_1 \wedge C_2])(y) = \max\{\beta \mid \beta \leq I(a_{C_1\wedge C_2}, y) \in \overline{\mathscr{A}}\}.$$

For direction ($\leq$), notice that $([C_1 \wedge C_2])(y) = \wedge_{b\in A}([\![C_1 \wedge C_2]\!](b) \to I(b,y))$. By the proof of the first claim above and the construction of the model, we have $([C_1 \wedge C_2])(y) = \wedge_{b\in A}(\max\{\alpha \mid \alpha \leq I(b,x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}\} \to \max\{\beta \mid \beta \leq I(b,y) \in \overline{\mathscr{A}}\})$. Suppose $\alpha_0 \leq ([C_1 \wedge C_2])(y)$. Therefore, we have $\alpha_0 \leq \max\{\alpha \mid \alpha \leq I(b,x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}\} \to \max\{\beta \mid \beta \leq I(b,y) \in \overline{\mathscr{A}}\}$ for every $b \in A$. By the creation rule, we have $1 \leq I(a_{C_1\wedge C_2}, x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}$, which means that $\max\{\alpha \mid \alpha \leq I(b,x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}\} = 1$. Therefore, we get $\alpha_0 \leq \max\{\beta \mid \beta \leq I(a_{C_1\wedge C_2}, y) \in \overline{\mathscr{A}}\}$. Hence, by completeness of $\mathbf{H}$, we have $([C_1 \wedge C_2])(y) \leq \max\{\beta \mid \beta \leq I(a_{C_1\wedge C_2}, y) \in \overline{\mathscr{A}}\}$.

For direction ($\geq$), suppose $\alpha_0 \leq I(a_{C_1\wedge C_2}, y)$. By construction of the model, there exists $\alpha_1 \in \mathbf{H}$, such that $\alpha_0 \leq \alpha_1$ and $\alpha_1 \leq I(a_{C_1\wedge C_2}, y) \in \overline{\mathscr{A}}$. Suppose $\alpha_2 \leq I(b,x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}$ for some $b \in A$, and $\alpha_2 \in \mathbf{H}$, by appending and basic rules, we get $\alpha_1 \wedge \alpha_2 \leq I(b,y) \in \overline{\mathscr{A}}$. Hence, we have $\alpha_1 \wedge \alpha_2 \leq \max\{\alpha \mid \alpha \leq I(b,y) \in \overline{\mathscr{A}}\}$ for any $\alpha_2 \leq I(b,x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}$. Therefore, by properties of $\mathbf{H}$, we have $\alpha_0 \leq \alpha_1 \leq \alpha_2 \to \alpha_1 \wedge \alpha_2 \leq \max\{\alpha \mid \alpha \leq I(b,x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}\} \to \max\{\beta \mid \beta \leq I(b,y) \in \overline{\mathscr{A}}\}$. Hence, by completeness of $\mathbf{H}$, we have $I(a_{C_1\wedge C_2}, y) \leq \max\{\alpha \mid \alpha \leq I(b,x_{C_1\wedge C_2}) \in \overline{\mathscr{A}}\} \to \max\{\beta \mid \beta \leq I(b,y) \in \overline{\mathscr{A}}\}$ for any $b \in A$. Therefore, we have $\max\{\beta \mid \beta \leq I(a_{C_1\wedge C_2}, y)\} \leq \wedge_{b\in A}([\![C_1 \wedge C_2]\!](b) \to I(b,y)) = ([C_1 \wedge C_2])(y)$.

2. The proof for $C = C_1 \vee C_2$ is similar to the previous one.

3. Suppose $C = [R_\Box]C_1$.

3.1. For the first claim, it is required to prove that

$$[\![[R_\Box]C_1]\!](b) = \max\{\beta \mid \beta \leq I(b,x_{\Box C_1}) \in \overline{\mathscr{A}}\}.$$

For direction ($\leq$), notice that $[\![[R_\Box]C_1]\!](b) = \bigwedge_{y\in X}(([C_1])(y) \to R_\Box(b,y))$. By induction and construction of the model, this is equivalent to $[\![[R_\Box]C_1]\!](b) = \bigwedge_{y\in X}(\max\{\alpha \mid \alpha \leq I(a_{C_1}, y)\} \to \max\{\beta \mid \beta \leq R_\Box(b,y) \in \overline{\mathscr{A}}\})$. Suppose $\alpha_0 \leq [\![[R_\Box]C_1]\!](b)$. By creation and basic rules, we have $1 \leq I(a_{C_1}, x_{C_1}) \in \overline{\mathscr{A}}$. By induction hypothesis claim above, we get $\alpha_0 \leq \max\{\beta \mid \beta \leq R_\Box(b,x_{C_1}) \in \overline{\mathscr{A}}\}$. By rule $R_\Box$, this implies $\alpha_0 \leq \max\{\beta \mid \beta \leq I(b,\Box x_{C_1}) = I(b,x_{\Box C_1}) \in \overline{\mathscr{A}}\}$. Therefore, $[\![[R_\Box]C_1]\!](b) \leq \max\{\beta \mid \beta \leq I(b,x_{\Box C_1}) \in \overline{\mathscr{A}}\}$.

For direction ($\geq$), suppose $\alpha_0 \leq I(b,x_{\Box C_1})$. By construction of the model, we have $\alpha_1 \leq I(b,x_{\Box C_1}) \in \overline{\mathscr{A}}$ for some $\alpha_0 \leq \alpha_1 \in \mathbf{H}$. Suppose $\alpha_2 \leq I(a_{C_1}, y) \in \overline{\mathscr{A}}$ for some y. Then by appending and $\Box$ rules, we get $\alpha_1 \wedge \alpha_2 \leq R_\Box(b,y)$. By properties of $\mathbf{H}$, we have $\alpha_0 \leq \alpha_1 \leq \alpha_2 \to \alpha_1 \wedge \alpha_2$. Hence we get $\alpha_0 \leq \max\{\alpha \mid \alpha \leq I(a_{C_1}, y)\} \to \max\{\beta \mid \beta \leq R_\Box(b,y) \in \overline{\mathscr{A}}\}$ for any y. Therefore, $\max\{\beta \mid \beta \leq I(b,x_{\Box C_1}) \in \overline{\mathscr{A}}\} \leq [\![[R_\Box]C_1]\!](b)$.

3.2. For the second claim, it is required to prove that

$$([[R_\Box C_1]])(y) = \max\{\beta \mid \beta \leq I(a_{\Box C_1}, y) \in \overline{\mathscr{A}}\}.$$

For direction ($\leq$), notice that $([[R_\Box]C_1])(y) = \wedge_{b\in A}([\![[R_\Box]C_1]\!](b) \to I(b,y))$. By the proof of the first claim and construction of the model, we have $([[R_\Box C_1]])(y) = \wedge_{b\in A}(\max\{\alpha \mid \alpha \leq I(b, x_{\Box C_1})\} \to \max\{\beta \mid \beta \leq I(b,y)\})$. Suppose $\alpha_0 \leq ([[R_\Box]C_1])(y)$. Then for every b, $\alpha_0 \leq \max\{\alpha \mid \alpha \leq I(b, x_{\Box C_1})\} \to \max\{\beta \mid \beta \leq I(b,y)\}$. By applying this to $1 \leq I(a_{\Box C_1}, x_{\Box C_1}) \in \overline{\mathscr{A}}$ added by creation rule, we get $\alpha_0 \leq \max\{\beta \mid \beta \leq I(a_{\Box C_1}, y)\}$. Therefore, $([[R_\Box C_1]])(y) \leq \max\{\beta \mid \beta \leq I(a_{\Box C_1}, y) \in \overline{\mathscr{A}}\}$.

For direction ($\geq$), suppose $\alpha_0 \leq I(a_{\Box C_1}, y)$. By construction of the model, there exists $\alpha_1 \in \mathbf{H}$ such that $\alpha_0 \leq \alpha_1$ and $\alpha_1 \leq I(a_{\Box C_1}, y) \in \overline{\mathscr{A}}$. Suppose $\alpha_2 \leq I(b, x_{\Box C_1}) \in \overline{\mathscr{A}}$ for some $b \in A$, $\alpha_2 \in \mathbf{H}$. Then, by appending and basic rule, we get $\alpha_1 \wedge \alpha_2 \leq I(b,y) \in \overline{\mathscr{A}}$. Therefore, for for any $\alpha_2 \leq I(b, x_{\Box C_1}) \in \overline{\mathscr{A}}$, we have $\alpha_1 \wedge \alpha_2 \leq \max\{\alpha \mid \alpha \leq I(b,y)\} \in \overline{\mathscr{A}}$. Hence, we have $\alpha_0 \leq \alpha_1 \leq \alpha_2 \to \alpha_1 \wedge \alpha_2 \leq \max\{\alpha \mid \alpha \leq I(b, x_{\Box C_1}) \in \overline{\mathscr{A}}\} \to \max\{\beta \mid \beta \leq I(b,y) \in \overline{\mathscr{A}}\}$. Therefore, by the completeness of $\mathbf{H}$, we have $I(a_{\Box C_1}, y) \leq \max\{\alpha \mid \alpha \leq I(b, x_{\Box C_1}) \in \overline{\mathscr{A}}\} \to \max\{\beta \mid \beta \leq I(b,y) \in \overline{\mathscr{A}}\}$. Therefore, we have $\max\{\beta \mid \beta \leq I(a_{\Box C_1}, y) \in \overline{\mathscr{A}}\} \leq ([[R_\Box C_1]])(y)$.

4. The proof for $C = \langle R_\Diamond \rangle C_1$ is similar to the previous one.

This concludes the proof. □

B Model for the first example knowledge base

In this section, we describe model for the first knowledge base defined in Section 4 obtained using unraveling and Tableaux Algorithm 2.

Let $\mathscr{K} = (\mathscr{A}, \mathscr{T})$ be a knowledge base such that $\mathscr{T} = \{C_4 \equiv C_1 \vee C_3, C_2 \equiv C_4 \wedge C_7, C_6 \equiv (C_1 \wedge C_2) \vee (C_2 \wedge C_3)\}$, and $\mathscr{A} = \{1 \leq P_1 : C_2, P_1 : C_6 < 1/2, 1 \leq y_1 :: [R_\Box]C_1, 1 \leq y_3 :: [R_\Box]C_3, P_2 R_\Box y_3 < 1, 1/2 \leq P_2 : C_1, 1/2 \leq y_1 :: C_1\}$. By unraveling TBox we get the following concept definitions.

1. $C_4 \equiv C_1 \vee C_3$
2. $C_2 \equiv (C_1 \vee C_3) \wedge C_7$
3. $C_6 \equiv (C_1 \wedge ((C_1 \vee C_3) \wedge C_7)) \vee (((C_1 \vee C_3) \wedge C_7) \wedge C_3) \equiv (C_1 \wedge C_7) \vee (C_7 \wedge C_3)$.

By substituting these definitions in ABox and running Algorithm 2, we get a model for $\mathscr{K}$ using construction described in Section 6. Table 1 lists symbols we use for different constants of the form a_C or x_C which appear in the model.

Table 2 lists all the object and feature constant names appearing in the model and value of I relation for every object-feature pair.

The relation $R_\Box$ is defined by $R_\Box(a_9, x_1) = R_\Box(a_9, x_3) = R_\Box(a_{10}, x_2) = R_\Box(a_{10}, x_3) = 0$, $R_\Box(a_9, y_1) = R_\Box(P_1, y_3) = 1/2$, and $R_\Box(b,y) = 0$ for any other b, y. The relation $R_\Diamond$ is defined by $R_\Diamond(y,b) = 0$ for any b, y. The model contains atomic concepts C_1, C_3, and C_7. For any of these concepts C, its interpretation is given by the tuple $(a_C^\uparrow, x_C^\downarrow)$. It is easy to verify that the relations $R_\Box$, and $R_\Diamond$ are I-compatible (In particular, it can be checked that Lemma 2 holds and the model defined above satisfies the knowledge base $\mathscr{K}$.

C Proof of inconsistency of the second example knowledge base

In this section, we show that the second example of knowledge base given in Section 4, i.e. knowledge base $\mathscr{K}$ with TBox axiom $C_5 \equiv C_1 \wedge C_3$, and with set of ABox axioms $\{1/2 \leq P_3 : [R_\Box]C_1, 1 \leq P_3 : [R_\Box]C_3, P_3 R_\Box y_4 < 1/2, 1/2 \leq y_4 :: C_5\}$ is inconsistent using tableaux algorithm.

a_1	a_{C_1}	x_1	x_{C_1}
a_2	a_{C_3}	x_2	x_{C_3}
a_3	$a_{C_1 \vee C_3}$	x_3	$x_{C_1 \vee C_3}$
a_4	$a_{(C_1 \vee C_3) \wedge C_7}$	x_4	$x_{(C_1 \vee C_3) \wedge C_7}$
a_5	a_{C_7}	x_5	x_{C_7}
a_6	$a_{C_1 \wedge C_7}$	x_6	$x_{C_1 \wedge C_7}$
a_7	$a_{C_3 \wedge C_7}$	x_7	$x_{C_3 \wedge C_7}$
a_8	$a_{(C_1 \wedge C_7) \vee (C_3 \wedge C_7)}$	x_8	$x_{(C_1 \wedge C_7) \vee (C_3 \wedge C_7)}$
a_9	$a_{\Box C_1}$	x_9	$x_{\Box C_1} = \Box x_{C_1}$
a_{10}	$a_{\Box C_3}$	x_{10}	$x_{\Box C_3} = \Box x_{C_3}$
a_{11}	$a_\top$	x_{11}	$x_\bot$

Table 1: symbols for constants of the form a_C or x_C appearing in the model

By unraveling, we replace any occurrence of C_5 in ABox with $C_1 \wedge C_3$. The following table shows terms added to the tableaux expansion of the resulting ABox.

Rule	**Premises**	**Added terms**
create		$1 \leq x_{C_1} :: C_1,\ 1 \leq x_{C_2} :: C_2$
$\Box$	$1/2 \leq P_3 : [R_\Box]C_1,\ 1 \leq x_{C_1} :: C_1$	$1/2 \leq P_3 R_\Box x_{C_1}$
$\Box$	$1 \leq P_3 : [R_\Box]C_3,\ 1 \leq x_{C_3} :: C_3$	$1 \leq P_3 R_\Box x_{C_3}$
$R_\Box$	$1/2 \leq P_3 R_\Box x_{C_1},\ 1 \leq P_3 R_\Box x_{C_3}$	$1/2 \leq \blacklozenge P_3 I x_{C_1},\ 1 \leq \blacklozenge P_3 I x_{C_3}$
x_C	$1/2 \leq \blacklozenge P_3 I x_{C_1},\ 1 \leq \blacklozenge P_3 I x_{C_3}$	$1/2 \leq \blacklozenge P_3 : C_3,\ 1 \leq \blacklozenge P_3 : C_1$
$\wedge_A$	$1/2 \leq \blacklozenge P_3 : C_3,\ 1 \leq \blacklozenge P_3 : C_1$	$1/2 \leq \blacklozenge P_3 : C_3 \wedge C_1$
create		$x_{C_1 \wedge C_3} :: C_1 \wedge C_3$
I	$x_{C_1 \wedge C_3} :: C_1 \wedge C_3,\ 1/2 \leq \blacklozenge P_3 : C_3 \wedge C_1$	$1/2 \leq \blacklozenge P_3 I x_{C_1 \wedge C_3}$
$\blacklozenge b$	$1/2 \leq \blacklozenge P_3 I x_{C_1 \wedge C_3}$	$1/2 \leq P_3 I \Box x_{C_1 \wedge C_3} = 1/2 \leq P_3 I x_{\Box(C_1 \wedge C_3)}$
x_C	$1/2 \leq P_3 I x_{\Box(C_1 \wedge C_3)}$	$1/2 \leq P_3 : [R_\Box](C_1 \wedge C_3)$
unravel	$1/2 \leq y_4 :: C_5$	$1/2 \leq y_4 :: C_1 \wedge C_3$
$\Box$	$1/2 \leq P_3 : [R_\Box](C_1 \wedge C_3),\ 1/2 \leq y_4 :: C_1 \wedge C_3$	$1/2 \leq P_3 R_\Box y_4$

The term $1/2 \leq P_3 R_\Box y_4$ clashes with the term $P_3 R_\Box y_4 < 1/2$ in $\mathscr{A}$. Hence proved.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	y_1	y_3	$\Box y_1$	$\Box y_3$	$\Box x_3$
a_1	1	0	1	0	0	0	0	0	0	0	0	1/2	0	0	0	0
a_2	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
a_3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
a_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
a_5	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
a_6	1	0	1	1	1	1	0	1	0	0	0	1/2	0	0	0	0
a_7	0	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0
a_8	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0
a_9	0	0	0	0	0	0	0	0	1	0	0	1	1/2	0	0	1
a_{10}	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1
$\blacklozenge a_9$	1	0	1	0	0	0	0	0	0	0	0	1/2	0	0	0	0
$\blacklozenge a_{10}$	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
a_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P_1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
P_2	1/2	0	1/2	1	1	0	0	0	0	0	0	1/2	0	0	1/2	0
$\blacklozenge P_2$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 2: Objects (A) and features (X) of model and Relation I between them

Regional, Lattice and Logical Representations of Neural Networks

Sandro Preto
Center for Mathematics, Computing and Cognition
Federal University of ABC, Brazil
Institute of Mathematics and Statistics
University of São Paulo, Brazil
sandro.preto@ufabc.edu.br

Marcelo Finger
Institute of Mathematics and Statistics
University of São Paulo, Brazil
mfinger@ime.usp.br

A possible path to the interpretability of neural networks is to (approximately) represent them in the regional format of piecewise linear functions, where regions of inputs are associated to linear functions computing the network outputs. We present an algorithm for the translation of feedforward neural networks with ReLU activation functions in hidden layers and truncated identity activation functions in the output layer. We also empirically investigate the complexity of regional representations outputted by our method for neural networks with varying sizes. Lattice and logical representations of neural networks are straightforward from regional representations as long as they satisfy a specific property. So we empirically investigate to what extent the translations by our algorithm satisfy such property.

1 Introduction

Neural networks are computational models that aim to generalize patterns found in datasets from which they are determined by means of a learning algorithm [8]. Despite the undeniable advancement in the state of the art of intelligent systems promoted by neural networks, their lack of interpretability is subject to criticism. Neural networks suffer from the *black box problem* due to the lack of justification for their results and the impossibility to directly inspect their learned information [3, 5].

As several architectures of neural networks realize piecewise linear functions or may be approximated by them, a path towards interpretability is through *regional format* representations of such neural networks and functions by explicit sets of pairs $\langle p, \Omega \rangle$ of a linear piece p and a region Ω such that, for a vector of input values $\mathbf{x} \in \Omega$, the output is given by $p(\mathbf{x})$. An algorithm for establishing regional representations from feedforward neural networks with rectified linear units as activation functions is proposed in [15].

The main goal of this work is to introduce an algorithm for computing regional format representations of *ReLU–TId neural networks*, which are feedforward neural networks with rectified linear units as activation functions in hidden layers and truncated identity as activation functions in the output layer. Such algorithm outputs representations in the *pre-closed regional format*, where regions are polyhedra. Rather than just adapting the iterative method in [15], we present a novel recursive approach that allows a correctness proof by a straightforward induction argument.

An important feature of neural networks is that they are compact representations of functions. Then, although regional representations might provide interpretability of neural networks, they also might be exponential in the size of their traditional representation as graphs. In Section 4, we empirically measure the complexity of regional representations determined by our method for randomly generated ReLU–TId neural networks with varying numbers of neurons and layers and varying layer sizes.

Cynthia Kop and Helida Salles Santos (Eds); 19th Workshop on
Logical and Semantic Frameworks with Applications (LSFA'24)
EPTCS 421, 2025, pp. 64–79, doi:10.4204/EPTCS.421.4

Lattice representation is another possibility for representing neural networks and is achieved by combining maximum and minimum operations over linear pieces. Such representations further enable the codification of ReLU–TId neural networks in logical systems as Łukasiewicz infinitely-valued logic ($Ł_\infty$) and its extensions [4, 7, 11, 12], leading to yet another path to interpretability. Lattice and logical representations find applications in the formal verification of neural networks in attempts to circumvent the black box problem and allow their use in critical tasks; for instance, in aircraft collision avoidance alerts and autonomous vehicles. There are methods for formal verification using the lattice representation of neural networks [1] and methods that codify properties of neural networks in the language of $Ł_\infty$ departing from their logical representation [13, 14].

Lattice and logical representations may be built in polynomial time from ReLU–TId neural networks given in the pre-closed regional format as long as such encodings satisfy the so-called *lattice property* (Section 2) [12]. In this case, the regional representations are said to be in the *closed regional format*. This work also aims at empirically experimenting how far from satisfying lattice property are randomly generated neural networks.

The rest of this work is organized as follows. Section 2 introduces neural networks and their graph, regional, lattice and logical representations. Section 3 presents an algorithm for translating ReLU–TId neural networks into the pre-closed regional format. Section 4 presents the results of experiments where we measure the complexity of representations in pre-closed regional format and their degree of satisfiability of lattice property.

2 Preliminaries: Some Neural Networks and Their Representations

Traditionally, a *feedforward neural network* N is given (and represented) by a graph whose nodes are partitioned into an ordered family of ordered sets $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, where each L_i is a *layer*. All nodes in layer L_i, for $i \in \{0, \ldots, \Lambda - 1\}$, are linked by an edge to all nodes in layer L_{i+1} establishing a computational circuit such that all output values of nodes in L_i are input values to each node in L_{i+1}. There is a linear function $f_j^i : \mathbb{R}^{|L_{i-1}|} \to \mathbb{R}$ associated to each node n_j^i in layer L_i, for $j \in \{1, \ldots, |L_i|\}$ and $i \in \{1, \ldots, \Lambda\}$. For a tuple of input values $\mathbf{x} = \langle x_1, \ldots, x_{|L_{i-1}|}\rangle \in \mathbb{R}^{|L_{i-1}|}$ to node n_j^i, it has as output the value $n_j^i(\mathbf{x}) = \rho_i \circ f_j^i(\mathbf{x})$, where $\rho_i : \mathbb{R} \to \mathbb{R}$ is an *activation function*. Thus, for $\mathbf{x} = \langle x_1, \ldots, x_{|L_{i-1}|}\rangle$ as input to layer L_i, it has as outputs the values in the tuple

$$L_i(\mathbf{x}) = \langle \rho_i \circ f_1^i(\mathbf{x}), \ldots, \rho_i \circ f_{|L_i|}^i(\mathbf{x})\rangle.$$

Input values to N in a tuple $\mathbf{x} = \langle x_1, \ldots, x_{|L_0|}\rangle \in \mathbb{R}^{|L_0|}$ are neatly associated to the nodes $n_1^0, \ldots, n_{|L_0|}^0 \in L_0$, called *input nodes*; thus, from such inputs, N produces the output values in the $|L_\Lambda|$-tuple

$$N(\mathbf{x}) = \langle N(\mathbf{x})_1, \ldots, N(\mathbf{x})_{|L_\Lambda|}\rangle = L_\Lambda \circ \cdots \circ L_1(\mathbf{x}).$$

Nodes in L_Λ are called *output nodes*. We say that each output node n_j^Λ in a neural network N computes a function for which the value $N(\mathbf{x})_j$ is given in function of the input values $\mathbf{x} \in \mathbb{R}^{|L_0|}$.

We might restrict the input values of a neural network to some set $R \subseteq \mathbb{R}$. In this work, we focus on *ReLU–TId neural networks*: they accept input values from $[0,1]$ and have as activation functions the *rectified linear unit* $\rho_i = \mathrm{ReLU} : \mathbb{R} \to \mathbb{R}$, given by $\mathrm{ReLU}(x) = \max(0, x)$, for $i \in \{1, \ldots, \Lambda - 1\}$, and the *truncated identity function* $\rho_\Lambda = \mathrm{TId} : \mathbb{R} \to \mathbb{R}$, given by $\mathrm{TId}(x) = \max(0, \min(1, x))$. Such activation

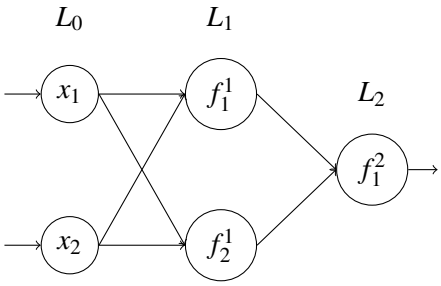


Figure 1: Graph of the ReLU–TId neural network E

functions may be given by piecewise linear definitions as follows:

$$\mathrm{ReLU}(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \qquad \mathrm{TId}(x) = \begin{cases} 0, & x < 0 \\ x, & 0 \leq x \leq 1 \\ 1, & x > 1 \end{cases} \tag{1}$$

Example 1 Let E be a ReLU–TId neural network with $\mathscr{L}_E = \{L_0, L_1, L_2\}$, where $L_0 = \{n_1^0, n_2^0\}$, $L_1 = \{n_1^1, n_2^1\}$ and $L_2 = \{n_1^2\}$. The graph of E depicted in Figure 1 highlights the input values x_1 and x_2 in layer L_0 and the functions $f_1^1, f_2^1, f_1^2 : \mathbb{R}^2 \to \mathbb{R}$ in layers L_1 and L_2, which are given by:

- $f_1^1(x_1, x_2) = \frac{4}{3}x_1 - x_2$;
- $f_2^1(x_1, x_2) = x_1 - x_2 + \frac{1}{2}$;
- $f_1^2(x_1, x_2) = x_1 + x_2 + \frac{1}{2}$.

For a tuple of inputs $\mathbf{e} = \langle \frac{1}{8}, \frac{1}{2} \rangle$ to E, we have $L_1(\mathbf{e}) = \langle \mathrm{ReLU}(-\frac{1}{3}), \mathrm{ReLU}(\frac{1}{8}) \rangle = \langle 0, \frac{1}{8} \rangle$ and, thus, $E(\mathbf{e}) = L_2(L_1(\mathbf{e})) = \mathrm{TId}(\frac{5}{8}) = \frac{5}{8}$. □

Before introducing another type of neural network, let us define a *rational McNaughton function* $f : [0,1]^n \to [0,1]$, which is a function that satisfies the following conditions:

- f is continuous with respect to the usual topology of $[0,1]$ as an interval of the real number line;
- There are linear polynomials $p_1, \ldots, p_m$ over $[0,1]^n$ with rational coefficients such that, for each point $\mathbf{x} \in [0,1]^n$, there is an index $i \in \{1, \ldots, m\}$ with $f(\mathbf{x}) = p_i(\mathbf{x})$. Polynomials $p_1, \ldots, p_m$ are the *linear pieces* of f.

A neural network whose $\nu = |L_\Lambda|$ output nodes exactly compute rational McNaughton functions in function of its input nodes is called a *ν-rational McNaughton neural network (ν-RMcN^3)*; a 1-RMcN^3 is also called *rational McNaughton neural network (RMcN^3)*.

A possibility to represent rational McNaughton functions (consequently, ν-rational McNaughton neural networks) is through the *regional formats* discussed in the following. Let Ω° denote the topological interior of $\Omega \subseteq [0,1]^n$ and say that, given functions $f, g : [0,1]^n \to [0,1]$, f is *above* g over the set Ω if $f(\mathbf{x}) \geq g(\mathbf{x})$, for all $\mathbf{x} \in \Omega$. A given rational McNaughton function $f : [0,1]^n \to [0,1]$ is said to be encoded in the *closed regional format* if it is given by m (not necessarily distinct) linear pieces

$$p_i(\mathbf{x}) = \gamma_{i0} + \gamma_{i1}x_1 + \cdots + \gamma_{in}x_n, \tag{2}$$

where $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in [0,1]^n$, $\gamma_{ij} \in \mathbb{Q}$ and $i \in \{1, \ldots, m\}$, such that each p_i is identical to f over a polyhedron $\Omega_i \subseteq [0,1]^n$, called *region*. These regions are determined by the finite intersection of half-spaces given by linear inequalities[1] as

$$\Omega_i = \left\{ \mathbf{x} \in [0,1]^n \;\middle|\; \omega_{j0} + \omega_{j1} x_1 + \cdots + \omega_{jn} x_n \geq 0,\ j \in \{1, \ldots, \lambda_{\Omega_i}\} \right\} \tag{3}$$

and such setting of linear pieces and regions satisfy the following properties:

- $\bigcup_{i=1}^{m} \Omega_i = [0,1]^n$;
- $\Omega_{i'}^{\circ} \cap \Omega_{i''}^{\circ} = \varnothing$, for $i' \neq i''$; and
- The *lattice property*: for $i \neq j$, there is k such that linear piece p_i is above linear piece p_k over region Ω_i and linear piece p_k is above linear piece p_j over region Ω_j.

Such an encoding is called *closed* because regions Ω_i are closed sets in the topological sense. As regions are given by such polyhedra described by (3), there is a polynomial procedure to establish whether a linear piece p is above q over region Ω: find the minimum value m of $p - q$ over Ω, which is a linear program and may be solved in polynomial time [2]; then, if $m \geq 0$, p is above q over Ω, otherwise, it is not.

The lattice property yields the possibility to represent rational McNaughton functions and ν-RMcN[3]s by *lattice representations*—i.e., based on operations of maximum and minimum over functions—as follows. First, let $f_{\Omega_j} : [0,1]^n \to [0,1]$ be the function given by

$$f_{\Omega_j}(\mathbf{x}) = \min \left\{ p_k(\mathbf{x}) \;\middle|\; p_k \text{ is above } p_j \text{ over } \Omega_j \right\}.$$

Note that $f_{\Omega_j}(\mathbf{x}) \leq f(\mathbf{x})$, for all $\mathbf{x} \in [0,1]^n$, which is obvious for $\mathbf{x} \in \Omega_j$ and follows from the lattice property for $\mathbf{x} \in \Omega_i$ where $i \neq j$. In this way, we have that

$$f(\mathbf{x}) = \max \left\{ f_{\Omega_j}(\mathbf{x}) \;\middle|\; j \in \{1, \ldots, m\} \right\}.$$

In [13], we find an example of the one-variable rational McNaughton function $f : [0,1] \to [0,1]$, whose graph is depicted in Figure 2. Function f has a lattice representation given by

$$f(x) = \max \left\{ f_{\Omega_1}(x),\ f_{\Omega_2}(x),\ f_{\Omega_3}(x),\ f_{\Omega_4}(x) \right\},$$

where $f_{\Omega_1}(x) = \min\{p_1(x), p_3(x)\}$, $f_{\Omega_2}(x) = f_{\Omega_3}(x) = \min\{p_2(x), p_3(x)\}$ and $f_{\Omega_4}(x) = \min\{p_2(x), p_4(x)\}$. Note that the lattice encoding just introduced may be employed in representing piecewise linear functions in general, not only rational McNaughton functions.

When a rational McNaughton function is given in an encoding that almost completely agrees with the closed regional format, with the sole exception that there is no guarantee that such encoding satisfies the lattice property (although it may still satisfy), we say that it is in the *pre-closed regional format*.

Unfortunately, lack of lattice property might entail the failure of lattice representation as in the following example taken from [12]. The rational McNaughton function f_E with graph in Figure 3a may have an encoding based on regions in Figure 3b; a linear piece p_i is associated to each region Ω_i. The dotted line in Figure 3b is the projection over $[0,1]^2$ of where p_3 intercepts p_5; note that such line passes

[1] We occasionally abuse notation by using the same symbol to refer both to a set of inequalities and to the polyhedron it determines.

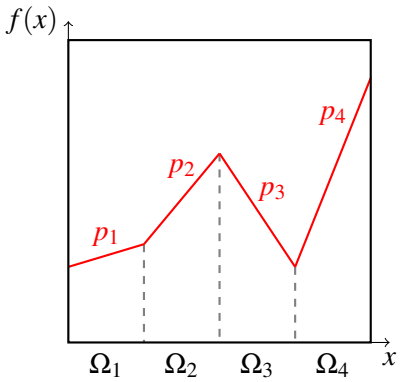
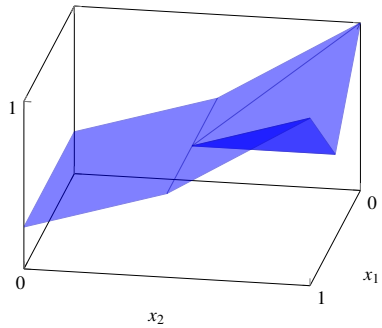
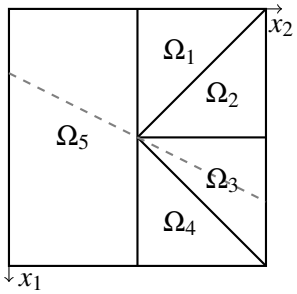


Figure 2: One-variable piecewise linear function



(a) Graph



(b) Region configuration

Figure 3: Function encoded in the pre-closed regional format

through the interior of both Ω_3 and Ω_5. There is no linear piece p_k such that p_3 is above p_k over Ω_3 and p_k is above p_5 over Ω_5. So an encoding for f_E based on regions Ω_1–Ω_5 may be at most encoded in pre-closed regional format.

Now, there is $\mathbf{x}_0 \in \Omega_3^\circ$ such that $p_5(\mathbf{x}_0) > p_3(\mathbf{x}_0)$. Therefore, $f_{\Omega_5}(\mathbf{x}_0) = p_5(\mathbf{x}_0) > p_3(\mathbf{x}_0) = \min\{p_1(\mathbf{x}_0), p_3(\mathbf{x}_0)\} = f_{\Omega_3}(\mathbf{x}_0)$, yielding that $\max\{f_{\Omega_j}(\mathbf{x}_0)\} > f_{\Omega_3}(\mathbf{x}_0)$, which eliminates the possibility of lattice representation. Such an issue may be circumvented by splitting region Ω_5, according to the dotted line in Figure 3b, in regions $\Omega_5' = \Omega_5 \cap \{p_5 - p_3 \geq 0\}$ and $\Omega_5'' = \Omega_5 \cap \{p_5 - p_3 \leq 0\}$. In general, repeatedly splitting a region according to projections of linear pieces intersections eventually achieves closed regional format [12, Theorem 7].

We may also represent rational McNaughton functions and ν-RMcN^{3}s in logical systems. For that, let us introduce the Łukasiewicz infinitely-valued logic ($Ł_\infty$). The basic language $\mathscr{L}$ of $Ł_\infty$ comprehends formulas freely generated from a countable set of propositional variables $\mathbb{P}$, a disjunction operator $\oplus$ and a negation operator $\neg$. A *valuation* is a function $v : \mathscr{L} \to [0,1]$, such that, for $\varphi, \psi \in \mathscr{L}$:

$$v(\varphi \oplus \psi) = \min(1, v(\varphi) + v(\psi)); \tag{4}$$

$$v(\neg\varphi) = 1 - v(\varphi). \tag{5}$$

From disjunction and negation we derive the following operators:

Conjunction:	$\varphi \odot \psi =_{\text{def}} \neg(\neg\varphi \oplus \neg\psi)$	$v(\varphi \odot \psi) = \max(0, v(\varphi) + v(\psi) - 1)$
Implication:	$\varphi \to \psi =_{\text{def}} \neg\varphi \oplus \psi$	$v(\varphi \to \psi) = \min(1, 1 - v(\varphi) + v(\psi))$
Maximum:	$\varphi \vee \psi =_{\text{def}} \neg(\neg\varphi \oplus \psi) \oplus \psi$	$v(\varphi \vee \psi) = \max(v(\varphi), v(\psi))$
Minimum:	$\varphi \wedge \psi =_{\text{def}} \neg(\neg\varphi \vee \neg\psi)$	$v(\varphi \wedge \psi) = \min(v(\varphi), v(\psi))$
Bi-implication:	$\varphi \leftrightarrow \psi =_{\text{def}} (\varphi \to \psi) \wedge (\psi \to \varphi)$	$v(\varphi \leftrightarrow \psi) = 1 - \lvert v(\varphi) - v(\psi)\rvert$

Note that, as lattice operations are expressed in Ł$_\infty$ by the minimum and maximum operators, piecewise linear functions might have a lattice representation in Ł$_\infty$ as far as their linear pieces are representable in this system. Indeed, the formulas of Ł$_\infty$ represent all the *McNaughton functions*, which are rational McNaughton functions constrained to allow only integer coefficients in their linear pieces [9, 10].

Unfortunately, Ł$_\infty$ cannot express rational McNaughton functions. For that, one possible path is to extend the language of Ł$_\infty$, which is done, for instance, by [7]. Another possibility is to implicitly represent such functions in plain Ł$_\infty$ using the technique of *representation modulo satisfiability*, which we introduce in the following [6, 11, 12].

Let us denote the Ł$_\infty$*-semantics*, that is the set of all valuations, by **Val**. Let us also denote by $\mathbf{Val}_\Phi$ the set of valuations $v \in \mathbf{Val}$ that satisfy a set of formulas Φ; we call such a restricted set of valuations a *semantics modulo satisfiability*. Given a rational McNaughton function $f : [0,1]^n \to [0,1]$, a formula φ_f and a set of formulas Φ_f, we say that φ_f *represents* f *modulo* Φ_f*-satisfiable* or that the pair $\langle \varphi_f, \Phi_f \rangle$ *represents* f *(in the system Ł$_\infty$-MODSAT)* if, for distinguished propositional variables $X_1, \ldots, X_n \in \mathbb{P}$:

- For all $\langle x_1, \ldots, x_n \rangle \in [0,1]^n$, there exists some valuation $v \in \mathbf{Val}_{\Phi_f}$, such that $v(X_i) = x_i$, for $i = 1, \ldots, n$;
- For all valuations $v, v' \in \mathbf{Val}_{\Phi_f}$ such that $v(X_i) = v'(X_i)$, for $i = 1, \ldots, n$, we have $v(\varphi_f) = v'(\varphi_f)$; and
- $f(v(X_1), \ldots, v(X_n)) = v(\varphi_f)$, for all $v \in \mathbf{Val}_{\Phi_f}$.

As an example, any constant function that takes value $\frac{1}{b}$, with $b \in \mathbb{N}^*$, may be represented by the pair

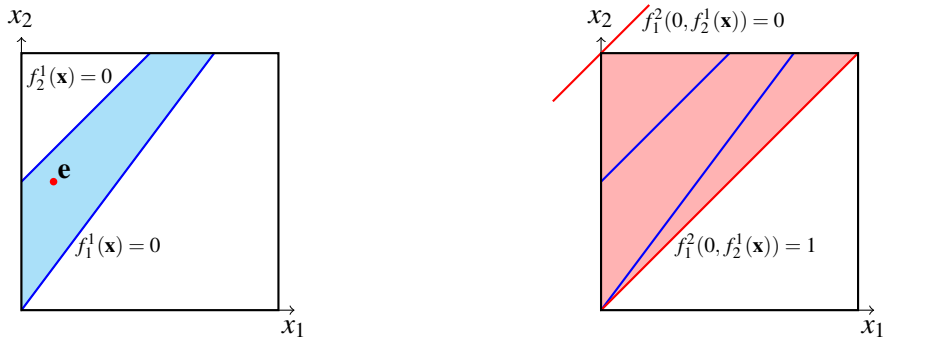
$$\langle \varphi, \Phi \rangle = \left\langle \; Z_{\frac{1}{b}}, \; \left\{ Z_{\frac{1}{b}} \leftrightarrow \neg(b-1) Z_{\frac{1}{b}} \right\} \; \right\rangle, \tag{6}$$

where formula φ is only the propositional variable $Z_{1/b}$ and set Φ is a singleton comprehending formula $Z_{1/b} \leftrightarrow \neg(b-1)Z_{1/b}$, which we denote by $\varphi_{1/b}$. In fact, for any valuation $v \in \mathbf{Val}_{\varphi_{1/b}} \neq \varnothing$, we have that $v(Z_{1/b}) = \frac{1}{b}$. Also, functions that take constant value $\frac{a}{b}$, with $a \in \mathbb{N}$, may be represented by the pair $\langle a\varphi, \Phi \rangle$.

Any rational McNaughton function may be represented in Ł$_\infty$-MODSAT. Moreover, there is a polynomial algorithm for the translation from a rational McNaughton function in closed regional format to its representation in such system [11, 12].

3 Neural Networks into Pre-Closed Regional Format

Given a ReLU–TId neural network N for which $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, we provide an algorithm to translate it into a tuple $\Xi_N = \langle \Xi_1, \ldots, \Xi_{|L_\Lambda|} \rangle$, where each Ξ_k, $k \in \{1, \ldots, |L_\Lambda|\}$, is the codification for a rational McNaughton function in pre-closed regional format, which we will show to be the function computed



(a) $f_1^1(\mathbf{x}) \leq 0$, $f_2^1(\mathbf{x}) \geq 0$ (b) $f_1^2(0, f_2^1(\mathbf{x})) \geq 0$, $f_1^2(0, f_2^1(\mathbf{x})) \leq 1$

Figure 4: Determining a region for neural network E

by N through the path to its k-th output node. Each Ξ_k is a set of pairs $\langle p, \Omega \rangle$, where p is a linear piece and Ω is its associated region.

Let us begin by analyzing the computation that takes place in each node of N. Given a tuple of input values $\mathbf{v} \in \mathbb{R}^{|L_{i-1}|}$ to node n_j^i of N, where $i \in \{1, \ldots, \Lambda - 1\}$ and so $\rho_i = \text{ReLU}$, the computation proceeds in two steps: first, the linear function $f_j^i(\mathbf{x})$ is evaluated for $\mathbf{x} = \mathbf{v}$; second, the activation function $\text{ReLU}(x)$ is evaluated for $x = f_j^i(\mathbf{v})$. In the second step, one from the two possible values highlighted in the piecewise definition of ReLU in (1) is chosen as the output of n_j^i. Such choice depends on the position of point $\mathbf{v} \in \mathbb{R}^{|L_{i-1}|}$ in relation to the hyperplane given by equation $f_j^i(\mathbf{x}) = 0$, since it may be a point lying either in the half-space given by $f_j^i(\mathbf{x}) \leq 0$ or in the half-space given by $f_j^i(\mathbf{x}) \geq 0$. For instance, for $\mathbf{e} = \langle \frac{1}{8}, \frac{1}{2} \rangle$ as input to the node n_1^1 of E in Example 1, the fact that $f_1^1(\mathbf{e}) \leq 0$ indicates that $\mathbf{e}$ lies in one of the half-spaces determined by $f_1^1(\mathbf{x}) = 0$ where, according to $\rho_1 = \text{ReLU}$, n_1^1 outputs 0. On the other hand, if $\mathbf{e}$ is given as input to the node n_2^1, as $f_2^1(\mathbf{e}) \geq 0$, $\mathbf{e}$ lies in the half-space determined by $f_2^1(\mathbf{x}) = 0$ where, according to $\rho_1 = \text{ReLU}$, n_2^1 outputs $f_2^1(\mathbf{e}) = \frac{1}{8}$. Figure 4a depicts the position of $\mathbf{e}$ in relation to these hyperplanes.

Similarly, in case $i = \Lambda$, we have that $\rho_i = \text{TId}$ and, then, one from three possible values is chosen for $\text{TId}(f_j^i(\mathbf{v}))$ depending on the position of $\mathbf{v} \in \mathbb{R}^{|L_{\Lambda-1}|}$ in relation to the hyperplanes $f_j^\Lambda(\mathbf{x}) = 0$ and $f_j^\Lambda(\mathbf{x}) = 1$. It may be a point lying either in the half-space given by $f_j^\Lambda(\mathbf{x}) \leq 0$, or in the half-space given by $f_j^\Lambda(\mathbf{x}) \geq 1$ or in the intersection of half-spaces $f_j^\Lambda(\mathbf{x}) \geq 0$ and $f_j^\Lambda(\mathbf{x}) \leq 1$. Again, for $\mathbf{e} = \langle \frac{1}{8}, \frac{1}{2} \rangle$ as input to the neural network E in Example 1, we have $\mathbf{e}_1 = L_1(\mathbf{e}) = \langle 0, \frac{1}{8} \rangle$. Since $f_1^2(\mathbf{e}_1) \geq 0$ and $f_1^2(\mathbf{e}_1) \leq 1$, $\mathbf{e}_1$ lies in a position relative to $f_1^2(\mathbf{x}) = 0$ and $f_1^2(\mathbf{x}) = 1$ where, according to $\rho_2 = \text{TId}$, n_1^2 outputs $f_1^2(\mathbf{e}_1) = \frac{5}{8}$.

Now, still considering Example 1, let $\mathbf{x} \in [0,1]^{|L_0|}$ be any point that, as $\mathbf{e} = \langle \frac{1}{8}, \frac{1}{2} \rangle$, satisfies both inequalities

$$f_1^1(\mathbf{x}) \leq 0 \quad \text{and} \quad f_2^1(\mathbf{x}) \geq 0. \tag{7}$$

Then, to $\mathbf{x}_1 = L_1(\mathbf{x})$ satisfy inequalities $f_1^2(\mathbf{x}_1) \geq 0$ and $f_1^2(\mathbf{x}_1) \leq 1$ is equivalent to $\mathbf{x}$ satisfy inequalities

$$f_1^2(0, f_2^1(\mathbf{x})) \geq 0 \quad \text{and} \quad f_1^2(0, f_2^1(\mathbf{x})) \leq 1. \tag{8}$$

As the former inequalities (7), the latter inequalities (8) are also linear over tuples from $[0,1]^{|L_0|}$ of input values to the neural network E. Moreover, for $\mathbf{x} \in [0,1]^{|L_0|}$ satisfying inequalities (7) and (8), we have

that $E(\mathbf{x}) = f_1^2(0, f_2^1(\mathbf{x}))$. In this way, we have just devised a region and its associated linear piece for the rational McNaughton function computed by neural network E.

Generalizing the observations above, the idea behind the base algorithm for building Ξ_k, for $k \in \{1,\dots,|L_\Lambda|\}$, is to compute each pair $\langle p, \Omega\rangle \in \Xi_k$ beginning by: associating a symbol between $\leq$ and $\geq$ to each node n_j^i, for $i < \Lambda$, alluding to one of the two possible positions of an input to n_j^i in relation to the hyperplane $f_j^i(\mathbf{x}) = 0$—i.e., lying in the half-space $f_j^i(\mathbf{x}) \leq 0$ or in the half-space $f_j^i(\mathbf{x}) \geq 0$—; and associating a symbol among $\leq$, $\geq$ and $\lesseqgtr$ to the node n_k^Λ alluding to one of the three possible positions of an input to n_k^Λ in relation to the hyperplanes $f_k^\Lambda(\mathbf{x}) = 0$ and $f_k^\Lambda(\mathbf{x}) = 1$—i.e., lying in the half-space $f_k^\Lambda(\mathbf{x}) \leq 0$, or in the half-space $f_k^\Lambda(\mathbf{x}) \geq 1$ or in the intersection of the half-spaces $f_k^\Lambda(\mathbf{x}) \geq 0$ and $f_k^\Lambda(\mathbf{x}) \leq 1$. These associations of symbols to all the nodes in layers $L_1,\dots,L_\Lambda$ determine a *configuration of symbols*. Then, the algorithm proceeds by defining $\Omega \subseteq [0,1]^{|L_0|}$ as an intersection of half-spaces based on such configuration of symbols and establishing a linear expression for p such that $N(\mathbf{x})_k = p(\mathbf{x})$, for $\mathbf{x} \in \Omega$.

Example 2 For the neural network E in Example 1, in a configuration of symbols where we associate $\leq$ to n_1^1 and $\geq$ to n_2^1, the consequent region Ω should comprehend the inequalities $\frac{4}{3}x_1 - x_2 \leq 0$ and $x_1 - x_2 + \frac{1}{2} \geq 0$ (shaded area in Figure 4a). For an input $\mathbf{x} \in [0,1]^2$ that satisfies these inequalities, we have the outputs $n_1^1(\mathbf{x}) = 0$ and $n_2^1(\mathbf{x}) = f_2^1(\mathbf{x})$ which, composed with f_1^Λ, gives us the expression $x_1 - x_2 + 1$. In this way, in case we complete the configuration of symbols by associating $\leq$ to n_1^Λ, Ω should comprehend the inequality $x_1 - x_2 + 1 \leq 0$ and p should be given by $p(x_1, x_2) = 0$. In case we associate $\geq$ to n_1^Λ, Ω should comprehend the inequality $x_1 - x_2 + 1 \geq 1$ and p should be given by $p(x_1, x_2) = 1$. In the last case, if we associate $\lesseqgtr$ to n_1^Λ, Ω should comprehend both inequalities $x_1 - x_2 + 1 \leq 1$ and $x_1 - x_2 + 1 \geq 0$ (shaded area in Figure 4b) and p should be given by $p(x_1, x_2) = x_1 - x_2 + 1$. Note that the last case is the only one where region Ω would be non-empty. □

In order to build the entire representative tuple $\Xi_N = \langle \Xi_1, \dots, \Xi_{|L_\Lambda|}\rangle$, the algorithm needs to compute all the pairs $\langle p, \Omega\rangle$, each one associated to a different configuration of symbols, for all possible configurations of symbols. Thus, the entire computation of Ξ_N ends up with $2^{|L_1|} \times \cdots \times 2^{|L_{\Lambda-1}|} \times 3|L_\Lambda|$ pairs $\langle p, \Omega\rangle$. Later, we introduce methods meant to be combined with the base algorithm that might circumvent such high complexity.

For establishing the base translation algorithm, we first fix some notation. Let κ_0^n and κ_1^n be the constant linear functions with domain $[0,1]^n$ and ranges equal to $\{0\}$ and $\{1\}$, respectively; and let $\chi_n : \{\leq, \geq\} \to \{\kappa_0^n, \kappa_1^n\}$ be the functions given by $\chi_n(\leq) = \kappa_0^n$ and $\chi_n(\geq) = \kappa_1^n$. Also, let $\pi_n^m : [0,1]^m \to \mathbb{R}$ be the projection functions given by $\pi_n^m(x_1,\dots,x_m) = x_n$, for $m \in \mathbb{N}$ and $1 \leq n \leq m$. The base translation algorithm is split into Algorithms 1 and 2.

Algorithm 1 treats the tuple $\Xi_N = \langle \Xi_1, \dots, \Xi_{|L_\Lambda|}\rangle$ as a variable to be updated as it runs; thus, it first sets each of the $\Xi_1, \dots, \Xi_{|L_\Lambda|}$ to the empty set as their initial values (lines 1 and 2). Then, it defines $\Omega_{[0,1]}$ as a set of inequalities common to all regions (line 3) and π as a tuple of projection functions (line 4), which will be suitable for compositions with functions f_j^1 related to the first layer L_1. It proceeds by calling the recursive routine NN2PWL-R($\Xi_N \parallel L_1, \Omega, \pi$) (line 5), where Ξ_N is an argument **passed by reference**, which means that whenever NN2PWL-R modifies the value of Ξ_N, it will also be modified in the scope of the calling function. Finally, Algorithm 1 returns Ξ_N with its final value (line 6).

Algorithm 2 describes the recursive routine NN2PWL-R that has as inputs a tuple $\Xi_N = \langle \Xi_1, \dots, \Xi_{|L_\Lambda|}\rangle$ to be updated, a ReLU–TId neural network N with a distinguished layer $L_i \in \mathscr{L}_N$, a set of inequalities Ω and a tuple of functions $f = \langle f_1, \dots, f_{|L_{i-1}|}\rangle$. If $L_i \neq L_\Lambda$, for all possible association of symbols $\leq$ and $\geq$ to the nodes $n_1^i, \dots, n_{|L_i|}^i$ summarized in the tuple of symbols $\bowtie = \langle \bowtie_1, \dots, \bowtie_{|L_i|}\rangle$, NN2PWL-R($\Xi_N \parallel L_i, \Omega, f$) proceeds by:

- Computing $\Omega^i_{\bowtie}$ as Ω extended by the half-spaces $f^i_j \circ f(\mathbf{x}) \bowtie_j 0$, for $j \in \{1,\ldots,|L_i|\}$, where $f = \langle f_1,\ldots,f_{|L_{i-1}|}\rangle$ is a tuple of linear functions such that $f(\mathbf{x}) = \langle f_1(\mathbf{x}),\ldots,f_{|L_{i-1}|}(\mathbf{x})\rangle = L_{i-1}\circ\cdots\circ L_1(\mathbf{x})$, for $\mathbf{x}\in\Omega$ (line 3);
- Computing the tuple of linear functions $f^i_{\bowtie}$ that is identical to the output of $L_i\circ\cdots\circ L_1$ for inputs $\mathbf{x}\in\Omega^i_{\bowtie}$, with assistance of functions $\chi_{|L_0|}$ (line 4);
- And calling itself again by NN2PWL-R$(\Xi_N \parallel L_{i+1}, \Omega^i_{\bowtie}, f^i_{\bowtie})$ (line 5).

If $L_i = L_\Lambda$, for each of the output nodes $n^\Lambda_1,\ldots,n^\Lambda_{|L_\Lambda|}$, NN2PWL-R$(\Xi_N \parallel L_i, \Omega, f)$ proceeds by:

- Computing $\Omega_{\leq}$, $\Omega_{\geq}$ and $\Omega_{\lesseqgtr}$ as Ω extended, respectively, by the half-space $f^\Lambda_k\circ f(\mathbf{x}) \leq 0$, the half-space $f^\Lambda_k\circ f(\mathbf{x}) \geq 1$ and the pair of half-spaces $f^\Lambda_k\circ f(\mathbf{x}) \geq 0$ and $f^\Lambda_k\circ f(\mathbf{x}) \leq 1$, where f is a tuple of linear functions such that $f(\mathbf{x}) = L_{\Lambda-1}\circ\cdots\circ L_1(\mathbf{x})$, for $\mathbf{x}\in\Omega$ (lines 9, 11 and 13);
- And rewriting Ξ_k by adding the pairs $\langle \kappa^{|L_0|}_0, \Omega_{\leq}\rangle$, $\langle f^\Lambda_k\circ f, \Omega_{\lesseqgtr}\rangle$ and $\langle \kappa^{|L_0|}_1, \Omega_{\geq}\rangle$ to it (lines 10, 12 and 14).

Let Ω be any region appearing in the output of the base algorithm; it is built in $\Lambda+1$ steps in a way that, in each step, new inequalities are added to a polyhedron (identified with a set of inequalities) until it becomes Ω. The first step adds the inequalities that determine $[0,1]^{|L_0|}$ (Algorithm 1, line 3). The next $\Lambda-1$ steps, where the produced polyhedra are named $\Omega^i_{\bowtie}$ (Algorithm 2, line 3), are associated to layers $L_1,\ldots,L_{\Lambda-1}$ of N. The final step, where the produced region Ω is named either as $\Omega_{\leq}$, $\Omega_{\geq}$ or $\Omega_{\lesseqgtr}$ (Algorithm 2, line 9, 11 or 13), is associated to layer L_Λ.

Algorithm 1 NN2PWL: puts neural networks in the closed regional format

Input: A ReLU–TId neural network N for which $\mathscr{L}_N = \{L_0,\ldots,L_\Lambda\}$.
Output: A set Ξ_N representing rational McNaughton functions computed by the output nodes of N.

1: $\Xi_1 := \varnothing, \ldots, \Xi_{|L_\Lambda|} := \varnothing$;
2: $\Xi_N := \langle \Xi_1,\ldots,\Xi_{|L_\Lambda|}\rangle$;
3: $\Omega_{[0,1]} := \{x_1 \geq 0, x_1 \leq 1, \ldots, x_{|L_0|} \geq 0, x_{|L_0|} \leq 1\}$;
4: $\pi := \langle \pi^{|L_0|}_1,\ldots,\pi^{|L_0|}_{|L_0|}\rangle$;
5: NN2PWL-R$(\Xi_N \parallel L_1, \Omega_{[0,1]}, \pi)$;
6: **return** Ξ_N;

Lemma 1 *Let a ReLU–TId neural network N, for which $\mathscr{L}_N = \{L_0,\ldots,L_\Lambda\}$, be given as input to Algorithm 1. Then, Algorithm 1 terminates and outputs a tuple $\Xi_N = \langle \Xi_1,\ldots,\Xi_{|L_\Lambda|}\rangle$ where, for $k\in\{1,\ldots,|L_\Lambda|\}$, we have:*

- $\bigcup_{\Omega,\ for\ \langle p,\Omega\rangle\in\Xi_k} \Omega = [0,1]^{|L_0|}$;
- $\Omega'^{\circ} \cap \Omega''^{\circ} = \varnothing$, *for distinct* $\langle p',\Omega'\rangle, \langle p'',\Omega''\rangle \in \Xi_k$. □

PROOF Algorithm 1 always terminates since all of its loops, which are originated from Algorithm 2 calls, range over some finite set and all recursive calls in Algorithm 2 increments the index of the input layer L_i, which will eventually reach the last layer L_Λ and break the recursion by falsifying the conditional statement in line 1 of Algorithm 2. Let Ξ_k be an entry in Ξ_N; all inequalities added to regions in Ξ_k determine half-spaces in $[0,1]^{|L_0|}$. Indeed, this is the case in the first step of the construction of regions (Algorithm 1, line 3). This is also the case for the remaining half-spaces, whose corresponding inequalities are recursively added in lines 3, 9, 11 and 13 of Algorithm 2 and depend on its input tuple of linear functions f, which, in turn, are inductively defined over $[0,1]^{|L_0|}$: first by π (Algorithm 1, line

Algorithm 2 NN2PWL-R: recursive routine called by NN2PWL

Input: A tuple $\Xi_N = \langle \Xi_1, \ldots, \Xi_{|L_\Lambda|} \rangle$, a ReLU-TId neural network N, for which $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, with a distinguished layer $L_i \neq L_0$, a set of inequalities Ω and a tuple of linear functions $f = \langle f_1, \ldots, f_{|L_{i-1}|} \rangle$.

1: **if** $L_i \neq L_\Lambda$ **then**
2: **for** $\bowtie \in \{\leq, \geq\}^{|L_i|}$ **do**
3: $\Omega^i_{\bowtie} := \Omega \cup \{f^i_j \circ f(\mathbf{x}) \bowtie_j 0 \mid j = 1, \ldots, |L_i|\}$;
4: $f^i_{\bowtie} := \langle \chi_{|L_0|}(\bowtie_1) \cdot (f^i_1 \circ f), \ldots, \chi_{|L_0|}(\bowtie_{|L_i|}) \cdot (f^i_{|L_i|} \circ f) \rangle$;
5: NN2PWL-R$(\Xi_N \parallel L_{i+1}, \Omega^i_{\bowtie}, f^i_{\bowtie})$;
6: **end for**
7: **else**
8: **for** $k = 1, \ldots, |L_\Lambda|$ **do**
9: $\Omega_{\leq} := \Omega \cup \{f^\Lambda_k \circ f(\mathbf{x}) \leq 0\}$;
10: $\Xi_k := \Xi_k \cup \{\langle \kappa_0^{|L_0|}, \Omega_{\leq} \rangle\}$;
11: $\Omega_{\leqslant} := \Omega \cup \{f^\Lambda_k \circ f(\mathbf{x}) \geq 0,\ f^\Lambda_k \circ f(\mathbf{x}) \leq 1\}$;
12: $\Xi_k := \Xi_k \cup \{\langle f^\Lambda_k \circ f, \Omega_{\leqslant} \rangle\}$;
13: $\Omega_{\geq} := \Omega \cup \{f^\Lambda_k \circ f(\mathbf{x}) \geq 1\}$;
14: $\Xi_k := \Xi_k \cup \{\langle \kappa_1^{|L_0|}, \Omega_{\geq} \rangle\}$;
15: **end for**
16: **end if**

4) in the first call of NN2PWL-R (Algorithm 1, line 5); then, by $f^i_{\bowtie}$, for $i \in \{2, \ldots, \Lambda\}$ (Algorithm 2, line 4) in the following $\Lambda - 1$ calls of NN2PWL-R (Algorithm 2, line 5). Let $\mathbf{x} \in [0,1]^{|L_0|}$; note that among the possibilities for inequalities to be added in each step of the construction of regions, there is certainly one that is satisfied by $\mathbf{x}$. Thus, with the suitable configuration of symbols, there is a region Ω of Ξ_k built such that $\mathbf{x} \in \Omega$. Now, in the construction of two regions Ω' and Ω'' of Ξ_k, with $\Omega' \neq \Omega''$, there is some step where the added inequalities differ for Ω' and Ω'' for the first time. Such differing inequalities guarantee that $\Omega'^{\circ} \cap \Omega''^{\circ} = \varnothing$, whether they appear in an intermediate step or the final one. ■

Lemma 2 *Let a ReLU–TId neural network N, for which $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, be given as input to Algorithm 1 and let Ξ_k be an entry in the outputted tuple Ξ_N for which $\langle p, \Omega \rangle \in \Xi_k$. If $\mathbf{x} \in \Omega$, then $N(\mathbf{x})_k = p(\mathbf{x})$.* □

PROOF Let $\mathbf{x} \in \Omega$. Note that $\Omega = \Omega_{[0,1]} \cap \Omega^1_{\bowtie} \cap \cdots \cap \Omega^{\Lambda-1}_{\bowtie} \cap \Omega_{\bowtie}$ and $\Omega_{[0,1]} \supseteq \Omega^1_{\bowtie} \supseteq \cdots \supseteq \Omega^{\Lambda-1}_{\bowtie} \supseteq \Omega_{\bowtie}$, where $\Omega^i_{\bowtie}$ is such that $\bowtie \in \{\leq, \geq\}^{|L_i|}$ and $\Omega_{\bowtie} \in \{\Omega_{\leq}, \Omega_{\leqslant}, \Omega_{\geq}\}$. Then, as $\mathbf{x} \in \Omega_{[0,1]}$, $\mathbf{x} \in [0,1]^{|L_0|}$. Also, as $\mathbf{x} \in \Omega^i_{\bowtie}$, for $i \in \{1, \ldots, \Lambda - 1\}$, the tuples of functions $f^i_{\bowtie}$ defined in line 4 of Algorithm 2, given as arguments in the recursive call of NN2PWL-R, are such that $f^i_{\bowtie}(\mathbf{x}) = L_i \circ \cdots \circ L_1(\mathbf{x})$. Indeed, as $\mathbf{x} \in \Omega^1_{\bowtie}$, $\mathbf{x}$ satisfies the inequalities

$$f^1_1(\mathbf{x}) = f^1_1 \circ \pi(\mathbf{x}) \bowtie_1 0, \quad \ldots, \quad f^1_{|L_1|}(\mathbf{x}) = f^1_{|L_1|} \circ \pi(\mathbf{x}) \bowtie_{|L_1|} 0.$$

Then, we have that

$$f^1_{\bowtie}(\mathbf{x}) = \langle \chi(\bowtie_1) \cdot f^1_1 \circ \pi(\mathbf{x}), \ldots, \chi(\bowtie_{|L_1|}) \cdot f^1_{|L_1|} \circ \pi(\mathbf{x}) \rangle = \langle \mathrm{ReLU}(f^1_1(\mathbf{x})), \ldots, \mathrm{ReLU}(f^1_{|L_1|}(\mathbf{x})) \rangle = L_1(\mathbf{x}).$$

Now, let us assume that $f^i_{\bowtie}(\mathbf{x}) = L_i \circ \cdots \circ L_1(\mathbf{x})$, for $\mathbf{x} \in \Omega$. As, in particular, $\mathbf{x} \in \Omega^{i+1}_{\bowtie}$, $\mathbf{x}$ satisfies the inequalities

$$f^{i+1}_1 \circ f^i_{\bowtie}(\mathbf{x}) = f^{i+1}_1 \circ L_i \circ \cdots \circ L_1(\mathbf{x}) \bowtie_1 0, \quad \ldots, \quad f^{i+1}_{|L_{i+1}|} \circ f^i_{\bowtie}(\mathbf{x}) = f^{i+1}_{|L_{i+1}|} \circ L_i \circ \cdots \circ L_1(\mathbf{x}) \bowtie_{|L_{i+1}|} 0,$$

it follows that

$$\begin{aligned} f_{\bowtie}^{i+1}(\mathbf{x}) &= \langle \chi(\bowtie_1) \cdot f_1^{i+1} \circ L_i \circ \cdots \circ L_1(\mathbf{x}), \quad \ldots, \quad \chi(\bowtie_{|L_1|}) \cdot f_{|L_1|}^{i+1} \circ L_i \circ \cdots \circ L_1(\mathbf{x}) \rangle \\ &= \langle \mathrm{ReLU}(f_1^{i+1} \circ L_i \circ \cdots \circ L_1(\mathbf{x})), \quad \ldots, \quad \mathrm{ReLU}(f_{|L_1|}^{i+1} \circ L_i \circ \cdots \circ L_1(\mathbf{x})) \rangle \\ &= L_{i+1} \circ \cdots \circ L_1(\mathbf{x}). \end{aligned}$$

Finally, in case $\Omega_{\bowtie} = \Omega_{\leqslant}$ (Algorithm 2, line 11), as, in particular, $\mathbf{x} \in \Omega_{\leqslant}$, we have that

$$0 \leq f_k^{\Lambda} \circ L_{\Lambda-1} \circ \cdots \circ L_1(\mathbf{x}) \leq 1.$$

Therefore,

$$N(\mathbf{x})_k = \mathrm{TId}(f_k^{\Lambda} \circ L_{\Lambda-1} \circ \cdots \circ L_1(\mathbf{x})) = f_k^{\Lambda} \circ L_{\Lambda-1} \circ \cdots \circ L_1(\mathbf{x}) = p(\mathbf{x}).$$

The other cases where Ω is either $\Omega_{<}$ or $\Omega_{>}$ are similar. ■

Theorem 1 (Correctness) *Let a ReLU–TId neural network N, for which $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, be given as input to Algorithm 1 and let $\Xi_N = \langle \Xi_1, \ldots, \Xi_n \rangle$ be its output. Then, each entry Ξ_k in Ξ_N codifies a rational McNaughton function in the pre-closed regional format which is exactly the function computed by N through the path to its k-th output node.* □

PROOF By construction and Lemma 1, regions in Ξ_k comply to the properties of pre-closed regional format. Lemma 2 establishes that evaluation via Ξ_k is the same as via the k-th output node. Since the function computed via the k-th output node is a composition of continuous functions—both linear functions associated to nodes of N and activation functions—, it is a continuous function. Therefore, entries in the tuple Ξ_N codify continuous functions which are rational McNaughton functions. ■

Corollary 1 *ReLU–TId neural networks are ν-rational McNaughton neural networks, where $\nu = |L_\Lambda|$.* □

3.1 Decreasing the execution time of the base algorithm

The base algorithm just introduced has the downside to be exponential in the number of nodes of a given neural network. For a neural network N with $\mathscr{L}_N = \{L_0, \ldots, L_\Lambda\}$, we have seen that it computes $3|L_\Lambda| \times 2^{|L_1|+\cdots+|L_{\Lambda-1}|}$ regions. However, many of such regions may be the empty set, which makes the outputs of the base algorithm examples of degenerate codification in pre-closed regional format.

Example 3 In a configuration of symbols where we associate $\geq$ to n_1^1 and $\leq$ to n_2^1 in the neural network E in Example 1, the corresponding inequalities $\frac{4}{3}x_1 - x - 2 \geq 0$ and $x_1 - x_2 + \frac{1}{2} \leq 0$ together, related to the first layer L_1, determine the empty set. □

In the step-by-step construction of a region $\Omega = \varnothing$ by the base algorithm, there is some step from the second when equations are added to the current polyhedron turning it into the empty set. In view of that, the first addition proposed for decreasing the execution time of the base algorithm consists in:

- Conditioning the call of NN2PWL-R in line 5 of Algorithm 2 by placing it within the scope of an if-statement that verifies whether $\Omega_{\bowtie}^i \neq \varnothing$;
- Conditioning the addition of new pairs $\langle p, \Omega_{\bowtie} \rangle$, for all $\bowtie \in \{\leq, \leqslant, \geq\}$, to tuple Ξ_N in lines 10, 12 and 14 of Algorithm 2 by placing these commands within the scope of if-statements that verify whether $\Omega_{<} \neq \varnothing$, $\Omega_{\leqslant} \neq \varnothing$ and $\Omega_{>} \neq \varnothing$.

Verifying whether $\Omega_{\bowtie} \neq \varnothing$ might significantly decrease the running time of the translation algorithm in practice. Indeed, each true statement $\Omega_{\bowtie} \neq \varnothing$ occurring in the the i-th step of the construction of regions, for $i \in \{1, \ldots, \Lambda - 1\}$, avoids a call of NN2PWL-R that, in the pure base algorithm, would yield

the computation of $3|L_\Lambda| \times 2^{|L_{i+1}|+\cdots+|L_{\Lambda-1}|}$ pairs $\langle p, \Omega \rangle$. On the other hand, verifying whether $\Omega_{\leq} \neq \varnothing$, $\Omega_{\lneq} \neq \varnothing$ or $\Omega_{\geq} \neq \varnothing$ in the last step of the construction of regions only prevents the algorithm to add pairs with empty regions to the regional format codification, which, nevertheless, makes the final representative tuple Ξ_N smaller.

A possible way to verify whether a polyhedron Ω given as in (3) is nonempty is by applying the known polynomial techniques used to verify whether a linear optimization program constrained by Ω is feasible [2].

For another method for easing the execution time of the base algorithm, observe that, for a layer L_i, for $i \in \{1, \ldots, \Lambda - 1\}$, each of the hyperplanes $f_j^i(\mathbf{x}) = 0$ related to nodes n_j^i of L_i, for $j \in \{1, \ldots, |L_i|\}$, divides the euclidean space $\mathbb{R}^{|L_0|}$ in two half-spaces determined by the inequalities $f_j^i(\mathbf{x}) \geq 0$ and $f_j^i(\mathbf{x}) \leq 0$. Each of these inequalities is added to half of the $2^{|L_i|}$ polyhedra generated in the first for-loop of Algorithm 2 (lines 2 to 6); these are the polyhedra generated in the i-th step of the construction of regions. Now, note that if the hyperplane $f_j^i(\mathbf{x}) = 0$ does not intercept the interior of the unit cube $[0,1]^{|L_0|}$, half of the new generated polyhedra are certainly empty. For instance, the hyperplane $x_1 + x_2 - 2 = 0$ does not intercept $[0,1]^{|L_0|}$. Thus, although the half-space given by $x_1 + x_2 - 2 \leq 0$ contains the entire unit cube $[0,1]^{|L_0|}$, the half-space given by $x_1 + x_2 - 2 \geq 0$ does not intersect it; so, if $x_1 + x_2 - 2 \geq 0$ is added to a polyhedron in some step of the construction of regions by the base algorithm, the regions generated from such polyhedron will be the empty set.

Thus, for the step related to layer L_i, for $i \in \{1, \ldots, \Lambda\}$, in the construction of regions, the proposed method consists in building a set $\mathbf{I} \subseteq \{\leq, \geq\}^{|L_i|}$ to be iterated instead of the set $\{\leq, \geq\}^{|L_i|}$ in the for-loop beginning in line 2 of Algorithm 2, so avoiding the generation of empty polyhedra. For that, we compute $\mathbf{I} = \mathbf{I}_1 \times \cdots \times \mathbf{I}_{|L_i|}$ where, for $j \in \{1, \ldots, |L_i|\}$,

$$\mathbf{I}_j = \begin{cases} \{\geq, \leq\}, & \text{if } f_j^i(\mathbf{x}) = 0 \text{ intercepts the interior of } [0,1]^{|L_0|} \\ \{\leq\}, & \text{if } f_j^i(\mathbf{x}) \leq 0 \text{ contains the entire } [0,1]^{|L_0|} \\ \{\geq\}, & \text{if } f_j^i(\mathbf{x}) \geq 0 \text{ contains the entire } [0,1]^{|L_0|} \end{cases}$$

Determining which is the case for each $\mathbf{I}_j$ may be done by solving both of the following maximization and minimization linear programs, which are known to be solvable in polynomial time [2]:

$$\begin{array}{ll} \max / \min & f_j^i(\mathbf{x}) \\ \text{subject to} & [0,1]^{|L_0|} \end{array}$$

Let M and m respectively be the maximum and the minimum optimum values of the linear programs above. Then: if $M \geq 0$ and $m \leq 0$ or if $M \leq 0$ and $m \geq 0$, $f_j^i(\mathbf{x}) = 0$ intercepts $[0,1]^{|L_0|}$; if $M \geq 0$ and $m \geq 0$, $f_j^i(\mathbf{x}) \geq 0$ contains $[0,1]^{|L_0|}$; and if $M \leq 0$ and $m \leq 0$, $f_j^i(\mathbf{x}) \leq 0$ contains $[0,1]^{|L_0|}$. The overall execution time of the translation algorithm, even with an additional routine for building $\mathbf{I}$, might be significantly smaller than the time for the original base algorithm. In fact, let $J \subseteq \{1, \ldots, |L_i|\}$ be the set of indexes such that $\mathbf{I}_j \neq \{\leq, \geq\}$ if, and only if, $j \in J$; then, the for-loop beginning in line 2 of Algorithm 2 has $2^{|L_i|-|J|}$ iterations instead of $2^{|L_i|}$.

Combining both of the methods described in this section with the base translation algorithm makes it compute exactly the same pairs $\langle p, \Omega \rangle$ that it would compute without such methods with the exception of the ones for which $\Omega = \varnothing$. Therefore, we are able to establish the following result.

Theorem 2 *Replacing the routine* NN2PWL-R *for a version of it that includes the methods proposed in this section maintains the correctness of Algorithm 1 established in Theorem 1.* □

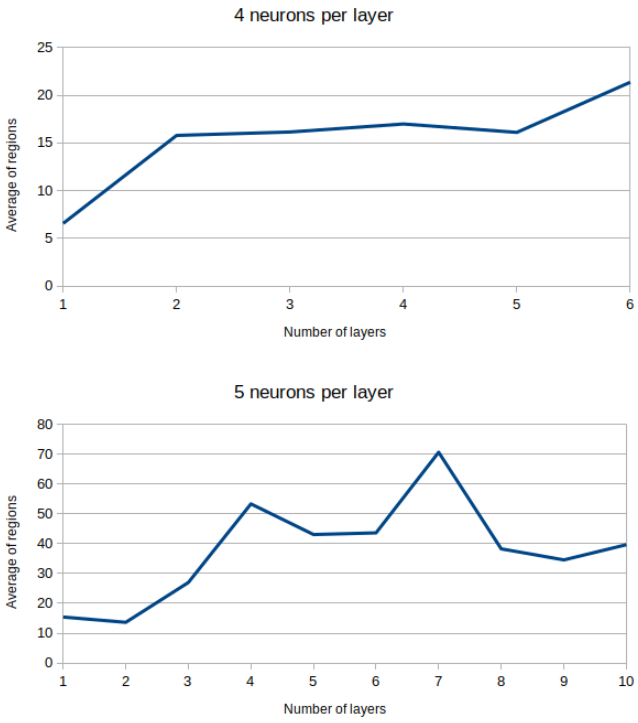


Figure 5: Experiments increasing the number of layers

4 Experiments and Results

We perform experiments for measuring the *complexity* of pre-closed regional format encodings of randomly generated ReLU–TId neural networks by counting the number of nonempty regions in them. All weights of the neural networks have the form $i+d$, where both i and d are uniformly generated from $\{-1,0,1\}$ and $[0,1)$, respectively.

For each encoding in pre-closed regional format, we also evaluate its *degree of satisfiability* of the lattice property by counting the number of pairs of regions $\langle \Omega_i, \Omega_j \rangle$ for which there is no linear piece p_k such that p_i is above p_k over Ω_i and p_k is above p_j over Ω_j, that is the number of pairs $\langle \Omega_i, \Omega_j \rangle$ that falsifies lattice property. If the counting is 0, such an encoding completely satisfies lattice property; the higher the count the further from satisfying lattice property the encoding is.

Implementations of NN2PWL, including the methods for decreasing its execution time, and a neural network generator were developed for the experiments; the source code is publicly available.[2]

In the first batch of experiments, for a fixed value h, ReLU–TId neural networks with h input neurons, h neurons in each hidden layer and one output neuron are generated. Such random generation is done in such a way that the neural networks are partitioned in L classes, each containing n neural networks with l hidden layers, for $l \in \{1,\ldots,L\}$. We ran such experiment for two parameter setups: $h=4$, $L=6$, $n=50$ and $h=5$, $L=10$, $n=25$. Figure 5 depicts the average number of regions extracted from the neural networks in each class of l hidden layers.

In order to analyze whether the results of the previous experiments depend on the distribution of

[2] `http://github.com/spreto/reluka`

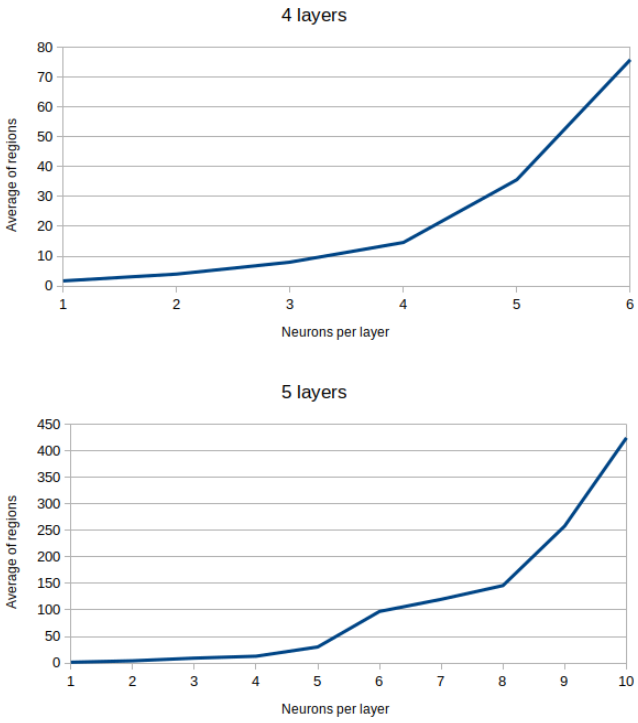


Figure 6: Experiments increasing the number of neurons per layer

neurons per layer, in the second batch of experiments, ReLU–TId neural networks with a fixed number l of hidden layers and one output neuron are generated. Now, the randomly generated neural networks are partitioned in M classes, each containing n neural networks with m input neurons and m neurons in each of their hidden layers, for $m \in \{1,\dots,M\}$. We ran such experiment for two parameter setups: $l = 4$, $M = 6$, $n = 50$ and $l = 5$, $M = 10$, $n = 25$. Figure 6 depicts the average number of regions extracted from the neural networks in each class of m neurons in the input layer and per hidden layer.

Note that the first experiment in both batches of experiments are related: for $l = m$, neural networks in a class with l layers (first experiment, first batch) have the same number of neurons than the neural networks in a class with m nodes per layer (first experiment, second batch). The same relation may be seen between the second experiments of each batch.

In all experiments, we may see that the average number of regions increases as long as the number of neurons increases. However, while such variation in the number of regions is smooth for varying the number of layers, a sharp variation may be perceived for varying the number of neurons per layer. A neural network with 5 hidden layers and 10 neurons in each of them (50 neurons in all hidden layers) achieved the maximum number of 1852 regions among all neural networks generated. For comparison, among the neural networks with 50 neurons distributed in 10 hidden layers (5 neurons per hidden layer), the maximum number of regions achieved is 228. And among all the neural networks with more than 5 hidden layers, but only 5 neurons in each of them, the maximum number of regions achieved is 446 (in a neural network with 7 hidden layers).

Regarding lattice property, among all 1100 ReLU–TId neural networks that were generated in all experiments, only one failed to satisfy it. Such neural network has 5 neurons in each of its 5 hidden layers (25 neurons in all hidden layers) and its pre-closed regional format encoding has 91 regions and

fails to fulfill lattice property for 36 pairs of regions $\langle \Omega_i, \Omega_j \rangle$.

5 Conclusions

We have proposed an algorithm for translating ReLU–TId neural networks into the pre-closed regional format, which is a more interpretable representation than the traditional graph one. We also proposed methods for decreasing the computation time of the base algorithm and proved that ReLU–TId neural networks are ν-rational McNaughton neural networks.

Empirically, we measured the complexity of pre-closed regional format encodings of randomly generated ReLU–TId neural networks by counting the number of nonempty regions in such encodings. We could verify a bigger increase in the number of regions in the encodings with wider, but fewer, layers than in the encodings with more, but thinner, layers. The fast increase of curves in Figure 6, related to the variation in the size of a fixed number of layers, points to the high complexity of regional representation. Therefore, the reported results foresee scaling issues in the regional representation of real-world neural networks, which often are larger than those generated in our investigation.

We have also investigated the degree of satisfiability of the lattice property by the neural networks generated in our experiments. The results empirically indicate that the outputs of NN2PWL lacking lattice property are a very rare event. Only one of the neural networks generated do not fulfill such a property.

For the future, approximate and less complex regional representations might be pursued. A possible path is to establish the reasonability of allowing encodings not satisfying lattice property as approximations of neural networks. From an exact perspective, one might investigate efficient procedures for turning a rational McNaughton function encoding in pre-closed regional format into closed regional format.

Funding

This work was carried out at the Center for Artificial Intelligence (C4AI-USP), with support by the São Paulo Research Foundation (FAPESP) [grant #2019/07665-4] and by the IBM Corporation. This study was financed in part by the São Paulo Research Foundation (FAPESP) [grants #2021/03117-2 to S.P., #2015/21880-4 and #2014/12236-1 to M.F.]; and the National Council for Scientific and Technological Development (CNPq) [grant PQ 303609/2018-4 to M.F.].

References

[1] Brendon G. Anderson, Samuel Pfrommer & Somayeh Sojoudi (2023): *Tight Certified Robustness via Min-Max Representations of ReLU Neural Networks*. In: *2023 62nd IEEE Conference on Decision and Control (CDC)*, pp. 6348–6355, doi:10.1109/CDC49753.2023.10383700.

[2] Dimitris Bertsimas & John N. Tsitsiklis (1997): *Introduction to linear optimization*. Athena scientific series in optimization and neural computation, Athena Scientific.

[3] Davide Castelvecchi (2016): *Can we open the black box of AI?* Nature 538(7623), pp. 20–23, doi:10.1038/538020a.

[4] Roberto L.O. Cignoli, Itala M.L. D'Ottaviano & Daniele Mundici (2000): *Algebraic Foundations of Many-Valued Reasoning*. Trends in Logic, Springer Netherlands, doi:10.1007/978-94-015-9480-6.

[5] Marcelo Finger (2020): *Logic in Times of Big Data*. In J. Acacio de Barros & Décio Krause, editors: *A True Polymath: A Tribute to Francisco Antonio Doria*, College Publications, pp. 184–198.

[6] Marcelo Finger & Sandro Preto (2020): *Probably Partially True: Satisfiability for Łukasiewicz Infinitely-Valued Probabilistic Logic and Related Topics*. Journal of Automated Reasoning 64(7), pp. 1269–1286, doi:10.1007/s10817-020-09558-9.

[7] Brunella Gerla (2001): *Rational Łukasiewicz Logic and DMV-algebras*. Neural Network World 11(6), pp. 579–594, doi:10.48550/arXiv.1211.5485

[8] Ian Goodfellow, Yoshua Bengio & Aaron Courville (2016): *Deep Learning*. MIT Press.

[9] R. McNaughton (1951): *A Theorem About Infinite-Valued Sentential Logic*. Journal of Symbolic Logic 16, pp. 1–13, doi:10.2307/2268660.

[10] Daniele Mundici (1994): *A constructive proof of McNaughton's theorem in infinite-valued logic*. The Journal of Symbolic Logic 59(2), pp. 596–602, doi:10.2307/2275410.

[11] Sandro Preto & Marcelo Finger (2020): *An Efficient Algorithm for Representing Piecewise Linear Functions into Logic*. Electronic Notes in Theoretical Computer Science 351, pp. 167–186, doi:10.1016/j.entcs.2020.08.009. Proceedings of LSFA 2020, the 15th International Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2020).

[12] Sandro Preto & Marcelo Finger (2022): *Efficient representation of piecewise linear functions into Łukasiewicz logic modulo satisfiability*. Mathematical Structures in Computer Science 32(9), pp. 1119–1144, doi:10.1017/S096012952200010X.

[13] Sandro Preto & Marcelo Finger (2023): *Effective Reasoning over Neural Networks Using Łukasiewicz Logic*. In Pascal Hitzler, Md Kamruzzaman Sarker & Aaron Eberhart, editors: Compendium of Neurosymbolic Artificial Intelligence, chapter 28, Frontiers in Artificial Intelligence and Applications 369, IOS Press, pp. 609–630, doi:10.3233/FAIA230160.

[14] Sandro Preto & Marcelo Finger (2023): *Proving properties of binary classification neural networks via Łukasiewicz logic*. Logic Journal of the IGPL 31(5), pp. 805–821, doi:10.1093/jigpal/jzac050.

[15] Haakon Robinson, Adil Rasheed & Omer San (2019): *Dissecting deep neural networks*. arXiv preprint arXiv:1910.03879, doi:10.48550/arXiv.1910.03879

Nominal Equational Rewriting and Narrowing

Mauricio Ayala-Rincón
University of Brasília, Brazil

Maribel Fernández
King's College London, UK

Daniele Nantes-Sobrinho
University of Brasília, Brazil
Imperial College London, UK

Daniella Santaguida*
University of Brasília, Brazil

Narrowing is a well-known technique that adds to term rewriting mechanisms the required power to search for solutions to equational problems. Rewriting and narrowing are well-studied in first-order term languages, but several problems remain to be investigated when dealing with languages with binders using nominal techniques. Applications in programming languages and theorem proving require reasoning modulo α-equivalence considering structural congruences generated by equational axioms, such as commutativity. This paper presents the first definitions of nominal rewriting and narrowing modulo an equational theory. We establish a property called nominal E-coherence and demonstrate its role in identifying normal forms of nominal terms. Additionally, we prove the nominal E-Lifting theorem, which ensures the correspondence between sequences of nominal equational rewriting steps and narrowing, crucial for developing a correct algorithm for nominal equational unification via nominal equational narrowing. We illustrate our results using the equational theory for commutativity.

1 Introduction

The nominal framework [15] has emerged as a promising approach for dealing with languages involving binders such as lambda calculus and first-order logic. In this framework, equality coincides with the α-equivalence relation, denoted as $\approx_\alpha$, and freshness constraints are integrated within the nominal reasoning rather than being relegated to the meta-language. For example, the expression $a\#M$ ("a is fresh for M") indicates that if a name a occurs in a term M, it must be abstracted by some binder, such as the λ in the lambda calculus, or $\exists, \forall$-quantification in first-order logic, i.e., a cannot occur free in M.

To enable reasoning within this framework, nominal unification [11, 22] was developed and formalised in proof assistants such as Isabelle [22], PVS [8] and Coq [4]. Nominal unification involves finding a substitution σ that solves the problem $s \stackrel{?}{\approx_?} t$, meaning $s\sigma \approx_\alpha t\sigma$, where s and t are nominal terms. It is well-known that unification is fundamental for automated reasoning, serving as the foundation for resolution-based proof assistants, type inference, and numerous other applications. While these applications are anticipated to extend to nominal unification, substantial work is required to verify this.

To pursue applications of the nominal framework, extensions of nominal unification with equational theories have been investigated. Initial efforts included integrating the theories of Associativity ($\approx_{\alpha,\mathsf{A}}$), Commutativity ($\approx_{\alpha,\mathsf{C}}$) and Associativity-Commutativity ($\approx_{\alpha,\mathsf{AC}}$) to α-equality [4]. Various algorithms for nominal unification modulo commutativity (C-unification) and formalisations of their correctness in proof assistants PVS and Coq have been developed [1, 2, 7, 5]. These development efforts reveal significant differences between first-order and nominal languages, such as the theory of C-unification, which has nullary unification type if α-equivalence is considered [1], contrasting with the finitary type of first-order C-unification [10].

*Author funded by Capes.

Cynthia Kop and Helida Salles Santos (Eds); 19th Workshop on Logical and Semantic Frameworks with Applications (LSFA'24)
EPTCS 421, 2025, pp. 80–97, doi:10.4204/EPTCS.421.5

Further investigations into nominal unification include exploring a *letrec* constructor and extensions involving atom variables [21]. Another example is the development of an algorithm for nominal C-matching [3], a special case of nominal C-unification (dealing with problems $s \stackrel{?}{\approx}^{\mathsf{C}}_{?} t$ where the substitution σ only applies in one side: $s\sigma \approx_{\alpha,\mathsf{C}} t$). Recently, a naive nominal extension of the Stickel-Fages first-order AC-unification algorithm introduced cyclicity in solutions produced by translations of unification problems to Diophantine systems as reported in [9], and this differs from the original (first-order) approach which has a terminating algorithm.

These developments underline the complexity of extending equational unification algorithms to the nominal framework, and new methods need to be proposed to obtain the desired extensions. An alternative approach to solving nominal unification problems modulo equational theories (i.e., nominal E-unification problems), developed in [6], involves the use of *nominal narrowing*[1]. This technique can be used when the equational theory E is presented by a convergent nominal rewriting system [14, 12]. Different extensions are needed for rewriting modulo E when such a presentation is impossible. However, nominal techniques modulo an equational theory E, and in particular, nominal E-rewriting, remain unexplored.

This work represents the first step towards developing nominal E-techniques, when using a convergent nominal rewrite system equivalent to the theory E is not possible. In first-order term languages [18, 23, 13], the standard technique is to split a set of identities T into a term rewriting system R and an equational part E, so that $\mathsf{T} = \mathsf{R} \cup \mathsf{E}$, considering the rewriting relation generated by R on the equivalence classes of terms generated by E. We propose extending this technique to nominal languages by adapting the notions of nominal rewriting [14, 20, 19] and nominal narrowing [6] to work modulo E, incorporating the relation $\approx_{\alpha,\mathsf{E}}$. These extensions result in the first definitions of nominal R/E-rewriting (Definition 3.1) and R,E-rewriting (Definition 3.2) as well as E-narrowing (Definition 3.8).

Nominal R/E-rewriting applies rules from R in the equivalence class modulo $\approx_{\alpha,\mathsf{E}}$ of a nominal term t, while R,E-rewriting uses nominal E-*matching* to determine if a rule in R applies to a nominal term, say t. This nominal term t may have variables, thus the definition of the relations R/E and R,E also feature freshness conditions. We prove that it is possible to identify the normal form of a nominal term, say t, with respect to the relation R/E (denoted $t \downarrow_{\mathsf{R}/\mathsf{E}}$) to the normal form of the same term, but with respect to the relation R,E (that is, $t \downarrow_{\mathsf{R},\mathsf{E}}$), when the relation R,E has a property called *nominal* E-*coherence*, whose extension from a corresponding property in first-order term language [17] is established here.

Proving the correspondence between sequences of nominal R,E-rewriting steps and E-narrowing steps (Nominal E-Lifting Theorem 4.6) is essential to develop an algorithm for nominal T-unification via nominal E-narrowing. Since the decidability of nominal T-unification relies on the decidability of nominal E-unification and E-matching, and so far, the only equational theory for which a nominal unification algorithm exists is commutativity C, a corollary of our developments is that the nominal C-Lifting Theorem holds. Finally, due to the volume of extensions that were necessary to establish nominal E-narrowing and rewriting, the final goal of using E-narrowing as a sound and complete procedure for solving nominal T-unification, remains ongoing work.

Summarising, our main contributions are:

1. We extend the definitions and concepts regarding rewriting modulo E to the nominal framework. For instance, we have nominal versions of the relations $\rightarrow_{\mathsf{R},\mathsf{E}}$ and $\rightarrow_{\mathsf{R}/\mathsf{E}}$ for rewriting, and $\rightsquigarrow_{\mathsf{R},\mathsf{E}}$ for nominal E-narrowing.

[1] Roughly, nominal narrowing is a generalisation of nominal rewriting by using nominal unification instead of nominal matching in its definition.

2. We prove technical auxiliary results relating $\to_{\mathsf{R},\mathsf{E}}$ and $\to_{\mathsf{R}/\mathsf{E}}$. These required the establishment of the nominal E-coherence property for R,E.
3. We prove the nominal E-Lifting Theorem (cf. Theorem 4.6) that establishes a correspondence between sequences of nominal E-narrowing $\rightsquigarrow_{\mathsf{R},\mathsf{E}}$ and nominal R,E-rewriting $\to_{\mathsf{R},\mathsf{E}}$.
4. Since C is the only equational theory for which a nominal unification algorithm exists, we illustrate our results using nominal R,C-rewriting and narrowing.

Organisation. In §2 we present the background necessary to read the paper. Novel material starts in §3, where we extend the notions of rewriting and narrowing modulo an equational theory E to the nominal framework and provide some examples. In §4 we present the classical Lifting Theorem, extended to the nominal framework, taking into account an equational E for which a nominal E-unification algorithm exists. §5 concludes the paper.

2 Preliminaries

While we assume the reader's familiarity with nominal techniques, we briefly recap some basic definitions. For more details, we refer to [14]. In this (and the following) section(s), we will use $\equiv$ for syntactic equality, $=$ for definitions and $\approx_\alpha$ for α-equality.

Syntax. Fix countable infinite pairwise disjoint sets of *atoms* $\mathbb{A} = \{a,b,c,\ldots\}$ and *variables* $\mathscr{X} = \{X,Y,Z,\ldots\}$. Atoms follow the *atom convention*: atoms $a,b,c,\ldots$ over $\mathbb{A}$ represent different names. Let Σ be a finite set of term-formers disjoint from $\mathbb{A}$ and $\mathscr{X}$ such that for each $f \in \Sigma$, a unique non-negative integer n (arity of f) is assigned. A *permutation* π is a bijection on $\mathbb{A}$ with finite domain, i.e., the set $\mathtt{dom}(\pi) = \{a \in \mathbb{A} \mid \pi(a) \neq a\}$ is finite. The identity permutation is denoted id. The composition of permutations π and π' is denoted $\pi \circ \pi'$ and π^{-1} denotes the inverse of the permutation π.

Nominal terms are defined inductively by the grammar:

$$s,t,u ::= a \mid \pi \cdot X \mid [a]t \mid f(t_1,\ldots,t_n),$$

where a is an *atom*, $\pi \cdot X$ is a (moderated/suspended) variable, $[a]t$ is the *abstraction* of a in the term t, and $f(t_1,\ldots,t_n)$ is a *function application* with $f \in \Sigma$ and $f : n$. We abbreviate $id \cdot X$ as X. A term is *ground* if it does not contain (moderated) variables. A *position* $\mathbb{C}$ is defined as a pair $(s,\_)$ of a term and a distinguished variable $\_ \in \mathscr{X}$ that occurs exactly once in s. We write $\mathbb{C}[s']$ for $\mathbb{C}[\_ \mapsto s']$ and if $s \equiv \mathbb{C}[s']$, we say that s' is a subterm of s with position $\mathbb{C}$. The root position will be denoted by $\mathbb{C} = [\_]$.

Remark 2.1 (Positions). Our definition of 'position' is equivalent to the standard notion of a point in the abstract syntax tree of a term, as defined, for example, in [10]. It is more convenient for us to identify this with the corresponding 'initial segment' of a nominal term, in which the 'hole' is a variable in $\mathscr{X}$; thus positions of a term can be expressed within our language.

A *permutation action* of π on a term t is defined by induction on the term structure as expected:

$$\pi \cdot a = \pi(a) \qquad \pi \cdot (\pi' \cdot X) = (\pi \circ \pi') \cdot X \qquad \pi \cdot [a]t = [\pi \cdot a](\pi \cdot t) \qquad \pi \cdot f(t_1,\ldots,t_n) = f(\pi \cdot t_1,\ldots,\pi \cdot t_n).$$

The *difference set* of two permutations $ds(\pi,\pi') := \{n \mid \pi \cdot n \neq \pi' \cdot n\}$. So $ds(\pi,\pi')\#X$ represents the set of constraints $\{n\#X \mid n \in ds(\pi,\pi')\}$. For example, if $\pi = (a\ b)(c\ d)$ and $\pi' = (c\ b)$, then $ds(\pi,\pi') =$

$$\frac{}{\Delta \vdash a\#b}\ (\#\text{ atom}) \qquad \frac{\Delta \vdash a\#t_1 \ \cdots\ \Delta \vdash a\#t_n}{\Delta \vdash a\#f(t_1,\cdots,t_n)}\ (\#\text{ app})$$

$$\frac{}{\Delta \vdash a\#[a]t}\ (\#\text{ a[a]}) \qquad \frac{\Delta \vdash a\#t}{\Delta \vdash a\#[b]t}\ (\#\text{ a[b]}) \qquad \frac{(\pi^{-1}\cdot a\#X)\in\Delta}{\Delta \vdash a\#\pi\cdot X}\ (\#\text{ var})$$

$$\frac{}{\Delta \vdash a \ \approx_\alpha\ a}\ (\approx_\alpha \text{ atom}) \qquad \frac{\Delta \vdash s_1 \ \approx_\alpha\ t_1 \ \cdots\ \Delta \vdash s_n \ \approx_\alpha\ t_n}{\Delta \vdash f(s_1,\cdots,s_n) \ \approx_\alpha\ f(t_1,\cdots,t_n)}\ (\approx_\alpha \text{ app})$$

$$\frac{\Delta \vdash s \ \approx_\alpha\ t}{\Delta \vdash [a]s \ \approx_\alpha\ [a]t}\ (\approx_\alpha \text{ [aa]}) \qquad \frac{\Delta \vdash s \ \approx_\alpha\ (a\ b)\cdot t \qquad \Delta \vdash a\#t}{\Delta \vdash [a]s \ \approx_\alpha\ [b]t}\ (\approx_\alpha \text{ [ab]})$$

$$\frac{ds(\pi,\pi')\#X \in \Delta}{\Delta \vdash \pi\cdot X \ \approx_\alpha\ \pi'\cdot X}\ (\approx_\alpha \text{ var})$$

Figure 1: Rules for # and $\approx_\alpha$

$\{a,b,c,d\}$ since π and π' act differently in each atom: note that $\pi(a)=b$ and $\pi'(a)=a$. In addition, $ds(\pi,\pi')\#X = \{a\#X, b\#X, c\#X, d\#X\}$.

A *substitution* θ is a mapping from a finite set of variables to terms. The *substitution action* $t\theta$ is defined as follows:

$$a\theta = a \qquad (\pi\cdot X)\theta = \pi\cdot(X\theta) \qquad ([a]t)\theta = [a](t\theta) \qquad f(t_1,\ldots,t_n)\theta = f(t_1\theta,\ldots,t_n\theta).$$

The domain of a substitution θ is written as $\mathtt{dom}(\theta)$, and the image is denoted as $\mathtt{Im}(\theta)$. Therefore, if $X \notin \mathtt{dom}(\theta)$ then $X\theta = X$. Also, if we restrict the domain to a certain set $V \subseteq \mathscr{X}$ of variables, we obtain the substitution $\theta|_V$, the *restriction of* θ *to* V. The identity substitution is denoted $\mathtt{Id}$. The composition of two substitutions θ_1 and θ_2 will be denoted by simple juxtaposition as $\theta_1\theta_2$ and it applies to a term as $t\theta_1\theta_2 = (t\theta_1)\theta_2$.

Nominal Constraints, Judgements and Rewriting. There are two kinds of constraints: $s \approx_\alpha t$ is an (alpha-)equality constraint and $a\#t$ is a freshness constraint which means that a cannot occur unabstracted in t. *Primitive constraints* have the form $a\#X$ and ∇,Δ denote finite sets of primitive constraints. We will use the abbreviation $a,b,c\#X$ to denote the set of freshness constraints $\{a\#X, b\#X, c\#X\}$. *Judgements* have the form $\Delta \vdash s \approx_\alpha t$ and $\Delta \vdash a\#t$ and are derived using the rules in Figure 1.

Given a finite set of freshness constraints Δ and a substitution θ, $\Delta\theta$ consists of the set of constraints $\{a\#X\theta \mid a\#X \in \Delta\}$ and $\langle\Delta\theta\rangle_{nf}$ consists of the set of freshness constraints obtained after applying the rules from Figure 1 in $\Delta\theta$, in a bottom-up manner. $\langle\Delta\theta\rangle_{nf}$ is *consistent* when it does not contain constraints of the form $a\#a$. A *problem* Pr is a set of constraints, and we write $\Delta \vdash Pr$ when for all $P \in Pr$ there is a derivation proof using the rules in Figure 1, taking elements of the context Δ as assumptions.

Example 2.1. Let $\Sigma_\lambda = \{\mathsf{lam}, \mathsf{app}\}$ denote the signature whose function symbols have arities $\mathsf{lam}:1$ and $\mathsf{app}:2$. Let $Pr = \mathsf{lam}[a]\mathsf{app}(a,X) \approx_\alpha \mathsf{lam}[b]\mathsf{app}(b,(a\ c)\cdot X)$ be a problem and $\Delta = \{a,b,c\#X\}$ be a context. We verify the derivability of $\Delta \vdash \mathsf{lam}[a]\mathsf{app}(a,X) \approx_\alpha \mathsf{lam}[b]\mathsf{app}(b,(a\ c)\cdot X)$:

$$\dfrac{\dfrac{\dfrac{}{\Delta \vdash a \approx_\alpha a}\,(\approx_\alpha \text{ atom}) \quad \dfrac{a,b,c\#X \in \Delta}{\Delta \vdash X \approx_\alpha (a\ b)(a\ c)\cdot X}\,(\approx_\alpha \text{ var})}{\Delta \vdash \mathsf{app}(a,X) \approx_\alpha \mathsf{app}(a,(a\ b)(a\ c)\cdot X)}\,(\approx_\alpha \text{ app}) \quad \dfrac{\dfrac{}{\Delta \vdash a\#b}\,(\#\text{ atom}) \quad \dfrac{c\#X \in \Delta}{\Delta \vdash a\#(a\ c)\cdot X}\,(\#\text{ var})}{\Delta \vdash a\#\mathsf{app}(b,(a\ c)\cdot X)}\,(\#\text{ app})}{\dfrac{\Delta \vdash [a]\mathsf{app}(a,X) \approx_\alpha [b]\mathsf{app}(b,(a\ c)\cdot X)}{\Delta \vdash \mathsf{lam}[a]\mathsf{app}(a,X) \approx_\alpha \mathsf{lam}[b]\mathsf{app}(b,(a\ c)\cdot X)}\,(\approx_\alpha \text{ app})}\,(\approx_\alpha \text{ [ab]})$$

A *term in context* $\Delta \vdash t$ expresses that the term t has the freshness constraints imposed by Δ. For example, $a\#X \vdash f(X,h(b))$ expresses that a cannot occur fresh in instances of X. Nominal rewriting rules can be defined under freshness constraints, i.e., $\nabla \vdash l \to r$ denotes a nominal rewriting rule. We denote by R, a finite set of nominal rewriting rules.

The *nominal rewriting relation* $\to_{\mathsf{R}}$ is defined as in [14]:

$$\frac{s \equiv \mathbb{C}[s'] \qquad \Delta \vdash \big(\nabla\theta, \qquad s' \approx_\alpha \pi\cdot(l\theta), \qquad \mathbb{C}[\pi\cdot(r\theta)] \approx_\alpha t\big)}{\Delta \vdash s \to_{\mathsf{R}} t}$$

for a substitution θ, a subterm s' of s, a position $\mathbb{C}$ and a nominal rule $\nabla \vdash l \to r \in \mathsf{R}$. We will omit the subscript R and write only $\Delta \vdash s \to t$ when there is no ambiguity.

Equality modulo an equational theory E. A nominal *identity* is a pair in context $\nabla \vdash (l,r)$ of nominal terms l and r under a (possibly empty) freshness context ∇. We denote such identity as $\nabla \vdash l \approx r$. A set E of identities induces an *equational theory*, which we will also denote as E.

The *nominal algebra equality modulo* E, denoted $\Delta \vdash s \approx_{\alpha,\mathsf{E}} t$, is the least transitive reflexive symmetric relation such that for any $(\nabla \vdash l \approx r) \in \mathsf{E}$, position $\mathbb{C}$, permutation π, substitution θ, and fresh context Γ (so if $a\#X \in \Gamma$ then a is not mentioned in Δ, s, t):

$$\frac{\Delta,\Gamma \vdash \big(\nabla\theta, \qquad s \approx_\alpha \mathbb{C}[\pi\cdot(l\theta)], \qquad \mathbb{C}[\pi\cdot(r\theta)] \approx_\alpha t\big)}{\Delta \vdash s \approx_{\alpha,\mathsf{E}} t}\,(Ax_{\mathsf{E}})$$

Remark 2.2. We can also define $\approx_{\alpha,\mathsf{E}}$ by extending the rules of Figure 1 with the dedicated rules for the identities defining E. For example, the identity expressing the commutativity of a function symbol f^{C} is $\mathsf{C} = \{\vdash f^{\mathsf{C}}(X,Y) \approx f^{\mathsf{C}}(Y,X)\}$. In this case, we need to add the following rule:

$$\frac{\Delta \vdash s_0 \ \approx_{\alpha,\mathsf{C}}\ t_i \qquad \Delta \vdash s_1 \ \approx_{\alpha,\mathsf{C}}\ t_{1-i} \qquad i = 0,1 \qquad f^{\mathsf{C}} \in \Sigma^{\mathsf{C}}}{\Delta \vdash f^{\mathsf{C}}(s_0,s_1) \ \approx_{\alpha,\mathsf{C}}\ f^{\mathsf{C}}(t_0,t_1)}\,(\approx_{\alpha,\mathsf{C}}\ \mathsf{C})$$

where Σ^{C} denotes a signature of commutative function symbols. Rule $(\approx_\alpha \text{ app})$ only applies when the function symbol f is not commutative. In addition, we need to modify the rules in Figure 1 to use $\approx_{\alpha,\mathsf{C}}$ instead of $\approx_\alpha$.

Note that if we define an equational theory E using the rule (Ax_{E}), the equational theory is a congruence relation, and $\vdash$ is compatible with substitutions by definition. However, this rule generates a lot of redundant derivations. To avoid this, we will use specific rules for each E, as for the commutative rule in Remark 2.2. This choice comes with a cost, now we need to prove the compatibility of $\vdash$ by substitution.

Definition 2.3 (E-Compatibility of $\vdash$ by substitutions)**.** An equational theory E is *compatible with* $\vdash$ *by substitutions* iff the following hold, whenever $\langle\Delta\theta\rangle_{nf}$ is consistent.

1. If $\Delta \vdash a\#t$ then $\langle\Delta\theta\rangle_{nf} \vdash a\#(t\theta)$.
2. If $\Delta \vdash s \approx_{\alpha,\mathsf{E}} t$ then $\langle\Delta\theta\rangle_{nf} \vdash (s\theta) \approx_{\alpha,\mathsf{E}} (t\theta)$.

3. If $\Delta \vdash Pr$ then $\langle \Delta\theta \rangle_{nf} \vdash Pr\theta$.

The next proposition guarantees the compatibility of judgements by substitutions when the theory C for commutativity is considered. This proposition is technical and will be used in the correspondence of one-step narrowing to one-step rewriting in Lemma 4.2.

Proposition 2.4. *The equational theory* C *is compatible with substitutions.*

Proof. By induction on the derivation of $\Delta \vdash a\#t$ or $\Delta \vdash s \approx_{\alpha,\mathsf{C}} t$, using the rules of Fig. 1 extended for $\approx_{\alpha,\mathsf{C}}$-equality. □

Nominal C-unification algorithm. We consider the rule-based algorithm for nominal C-unification, introduced in [2] and defined by the rules presented in Figure 2. The rules act on triples $\mathscr{P} = (\Delta, \theta, Pr)$, where Δ is a freshness context, θ is a substitution and Pr is a C-problem, i.e., a set of freshness and $\approx_{\alpha,\mathsf{C}}$-equality constraints. We will denote the triples by $\mathscr{P}, \mathscr{Q}, \mathscr{S}, \cdots$.

Definition 2.5. (C-solution) A C-*solution* for a triple $\mathscr{P} = (\Delta, \delta, Pr)$ is a pair (Δ', θ) where the following conditions are satisfied:

1. $\Delta' \vdash \Delta\theta$;
2. $\Delta' \vdash a\#t\theta$, if $a\#t \in Pr$;
3. $\Delta' \vdash s\theta \approx_{\alpha,\mathsf{C}} t\theta$, if $s \approx_{\alpha,\mathsf{C}} t \in Pr$;
4. there is a substitution θ' such that $\Delta' \vdash \delta\theta' \approx_{\alpha,\mathsf{C}} \theta$.

If there is no (Δ', θ), then we say that the problem $\mathscr{P}$ is *unsolvable*. Also $\mathscr{U}_{\mathsf{C}}(\mathscr{P})$ denotes the set of all C-solutions of the triple $\mathscr{P}$.

Let (Δ_1, θ_1) and (Δ_2, θ_2) be solutions in $\mathscr{U}_{\mathsf{C}}(\mathscr{P})$. We say that (Δ_1, θ_1) is *more general than* (Δ_2, θ_2), and denote it as $(\Delta_1, \theta_1) \leq_{\mathsf{C}} (\Delta_2, \theta_2)$, if there exists a substitution θ' such that $\Delta_2 \vdash X\theta_1\theta' \approx_{\alpha,\mathsf{C}} X\theta_2$, for all $X \in \mathscr{X}$ and $\Delta_2 \vdash \Delta_1\theta'$. We write $\leq_{\mathsf{C}}^{V}$ for the restriction of $\leq_{\mathsf{C}}$ to a set V of variables.

Definition 2.6. (Nominal C-unification problem) A *nominal* C-*unification problem (in context)* is a pair $(\nabla \vdash l) \stackrel{\mathsf{C}}{\approx}_? (\Delta \vdash s)$. The pair (Δ', θ) is an C-solution, or C-unifier, of $(\nabla \vdash l) \stackrel{\mathsf{C}}{\approx}_? (\Delta \vdash s)$ iff (Δ', θ) is a C-solution of the triple $\mathscr{P} = (\{\nabla, \Delta\}, \mathtt{Id}, \{l \approx_{\alpha,\mathsf{C}} s\})$, that is, conditions (1)-(4) of Definition 2.5 are satisfied. $\mathscr{U}_{\mathsf{C}}(\nabla \vdash l, \Delta \vdash s)$ denotes the set of all C-solutions of $(\nabla \vdash l) \stackrel{\mathsf{C}}{\approx}_? (\Delta \vdash s)$. If ∇ and Δ are empty we write simply $\mathscr{U}_{\mathsf{C}}(l, s)$. A subset $\mathscr{V} \in \mathscr{U}_{\mathsf{C}}(\mathscr{P})$ is said to be a *complete set of* C-*solutions of* $\mathscr{P}$ if for all $(\Delta_1, \theta_1) \in \mathscr{U}_{\mathsf{C}}(\mathscr{P})$, there exists $(\Delta_2, \theta_2) \in \mathscr{V}$ such that $(\Delta_2, \theta_2) \leq_{\mathsf{C}} (\Delta_1, \theta_1)$.

The following example illustrates the use of the nominal C-unification algorithm to solve a nominal C-unification problem.

Example 2.2. Let $\Sigma = \{h : 1, f^{\mathsf{C}} : 2, \oplus : 2\}$ be a signature, where f^{C} and $\oplus$ are commutative symbols, i.e., and $\mathsf{C} = \{ \vdash f^{\mathsf{C}}(X,Y) \approx f^{\mathsf{C}}(Y,X), \vdash X \oplus Y \approx Y \oplus X\}$ be the axioms defining the theory. Consider the C-unification problem $(\emptyset \vdash h(Y)) \stackrel{\mathsf{C}}{\approx}_? (\emptyset \vdash h(f^{\mathsf{C}}([b][a]X, X)))$ which has the associated triple $(\emptyset, \mathtt{Id}, \{h(Y) \stackrel{\mathsf{C}}{\approx}_? h(f^{\mathsf{C}}([b][a]X, X))\})$. By applying the rules from Figure 2 we get the following:

$$
\begin{aligned}
(\emptyset, \mathtt{Id}, \{h(Y) \stackrel{\mathsf{C}}{\approx}_? h(f^{\mathsf{C}}([b][a]X, X))\}) &\Longrightarrow_{(\approx_{\alpha,\mathsf{C}}\ \text{app})} (\emptyset, \mathtt{Id}, \{Y \stackrel{\mathsf{C}}{\approx}_? f^{\mathsf{C}}([b][a]X, X)\}) \\
&\Longrightarrow_{(\approx_{\alpha,\mathsf{C}}\ \text{inst})} (\emptyset, \theta_0 = [Y \mapsto f^{\mathsf{C}}([b][a]X, X)], \{f^{\mathsf{C}}([b][a]X, X) \stackrel{\mathsf{C}}{\approx}_? f^{\mathsf{C}}([b][a]X, X)\}) \\
&\Longrightarrow_{(\approx_{\alpha,\mathsf{C}}\ \text{refl})} (\emptyset, \theta_0 = [Y \mapsto f^{\mathsf{C}}([b][a]X, X)], \emptyset)
\end{aligned}
$$

$$
\begin{array}{lrcl}
\text{(\# ab)} & (\Delta, \theta, Pr \uplus \{a\#b\}) & \Longrightarrow & (\Delta, \theta, Pr) \\
\text{(\# app)} & (\Delta, \theta, Pr \uplus \{a\#f(t_1, \cdots, t_n)\}) & \Longrightarrow & (\Delta, \theta, Pr \cup \{a\#t_1, \cdots, a\#t_n\}) \\
\text{(\# a[a])} & (\Delta, \theta, Pr \uplus \{a\#[a]t\}) & \Longrightarrow & (\Delta, \theta, Pr) \\
\text{(\# a[b])} & (\Delta, \theta, Pr \uplus \{a\#[b]t\}) & \Longrightarrow & (\Delta, \theta, Pr \cup \{a\#t\}) \\
\text{(\# var)} & (\Delta, \theta, Pr \uplus \{a\#\pi \cdot X\}) & \Longrightarrow & (\{(\pi^{-1} \cdot a)\#X\} \cup \Delta, \theta, Pr)
\end{array}
$$

$$
\begin{array}{lrcl}
(\approx_{\alpha,\mathsf{C}} \text{refl}) & (\Delta, \theta, Pr \uplus \{s \approx_{\alpha,\mathsf{C}} s\}) & \Longrightarrow & (\Delta, \theta, Pr) \\
(\approx_{\alpha,\mathsf{C}} \text{app}) & (\Delta, \theta, Pr \uplus \{f(\bar{s})_n \approx_{\alpha,\mathsf{C}} f(\bar{t})_n\}) & \Longrightarrow & (\Delta, \theta, Pr \cup \bigcup \{s_i \approx_{\alpha,\mathsf{C}} t_i\}) \\
(\approx_{\alpha,\mathsf{C}} \text{C}) & (\Delta, \theta, Pr \uplus \{f^{\mathsf{C}} s \approx_{\alpha,\mathsf{C}} f^{\mathsf{C}} t\}) & \Longrightarrow & (\Delta, \theta, Pr \cup \{s \approx_{\alpha,\mathsf{C}} v\}), \text{where } \; s = (s_0, s_1) \\
 & & & \text{and } \; t = (t_0, t_1),\, v = (t_i, t_{(1-i)}), i = 0, 1 \\
(\approx_{\alpha,\mathsf{C}} \text{[aa]}) & (\Delta, \theta, Pr \uplus \{[a]s \approx_{\alpha,\mathsf{C}} [a]t\}) & \Longrightarrow & (\Delta, \theta, Pr \cup \{s \approx_{\alpha,\mathsf{C}} t\}) \\
(\approx_{\alpha,\mathsf{C}} \text{[ab]}) & (\Delta, \theta, Pr \uplus \{[a]s \approx_{\alpha,\mathsf{C}} [b]t\}) & \Longrightarrow & (\Delta, \theta, Pr \cup \{s \approx_{\alpha,\mathsf{C}} (a\ b) \cdot t, a\#t\}) \\
(\approx_{\alpha,\mathsf{C}} \text{inst}) & (\Delta, \theta, Pr \uplus \{\pi \cdot X \approx_{\alpha,\mathsf{C}} t\}) & \Longrightarrow & (\Delta, \theta', Pr[X \mapsto \pi^{-1} \cdot t] \cup \bigcup_{\substack{Y \in dom(\theta'), \\ a\#Y \in \Delta}} \{a\#Y\theta'\}), \\
 & & & \text{let } \; \theta' := \theta[X \mapsto \pi^{-1} \cdot t], \\
 & & & \text{if } \; X \notin Var(t) \\
(\approx_{\alpha,\mathsf{C}} \text{inv}) & (\Delta, \theta, Pr \uplus \{\pi \cdot X \approx_{\alpha,\mathsf{C}} \pi' \cdot X\}) & \Longrightarrow & (\Delta, \theta, Pr \cup \{(\pi')^{-1} \circ \pi \cdot X \approx_{\alpha,\mathsf{C}} X\}) \\
 & & & \text{if } \; \pi' \neq \mathtt{Id}
\end{array}
$$

Figure 2: Simplification rules for # and $\approx_{\alpha,\mathsf{C}}$. $\uplus$ denotes disjoint union

Thus, we get the C-solution $(\emptyset, \theta_0)$.

Now consider the C-unification problem $(\emptyset \vdash f^{\mathsf{C}}([a][b]Z, Z)) \stackrel{\mathsf{C}}{\approx}_? (\emptyset \vdash f^{\mathsf{C}}([b][a]X, X))$, which has the associated triple $(\emptyset, \mathtt{Id}, \{f^{\mathsf{C}}([a][b]Z, Z)) \stackrel{\mathsf{C}}{\approx}_? f^{\mathsf{C}}([b][a]X, X)\})$. Using the Nominal C-unification algorithm we get the following:

$$
\begin{array}{l}
(\emptyset, \mathtt{Id}, \{f^{\mathsf{C}}([a][b]Z, Z)) \stackrel{\mathsf{C}}{\approx}_? f^{\mathsf{C}}([b][a]X, X)\}) \quad \Longrightarrow_{(\approx_{\alpha,\mathsf{C}}\ \text{C})} \\
\qquad \Longrightarrow_{(\approx_{\alpha,\mathsf{C}}\ \text{C})} (\emptyset, \mathtt{Id}, \{[a][b]Z \stackrel{\mathsf{C}}{\approx}_? [b][a]X, Z \stackrel{\mathsf{C}}{\approx}_? X\}) \\
\qquad \Longrightarrow_{(\approx_{\alpha,\mathsf{C}}\ \text{inst})} (\emptyset, \theta_1 = [Z \mapsto X], \{[a][b]X \stackrel{\mathsf{C}}{\approx}_? [b][a]X, X \stackrel{\mathsf{C}}{\approx}_? X\}) \\
\qquad \Longrightarrow_{(\approx_{\alpha,\mathsf{C}}\ \text{refl})} (\emptyset, \theta_1, \{[a][b]X \stackrel{\mathsf{C}}{\approx}_? [b][a]X\}) \\
\qquad \Longrightarrow_{(\approx_{\alpha,\mathsf{C}}\ \text{[ab]})} (\emptyset, \theta_1, \{[b]X \stackrel{\mathsf{C}}{\approx}_? (a\ b) \cdot [a]X, a\#[a]X\}) \\
\qquad \Longrightarrow_{(\#\ \text{a[a]})} (\emptyset, \theta_1, \{[b]X \stackrel{\mathsf{C}}{\approx}_? [b](a\ b) \cdot X\}) \\
\qquad \Longrightarrow_{(\approx_{\alpha,\mathsf{C}}\ \text{[bb]})} (\emptyset, \theta_1, \{X \stackrel{\mathsf{C}}{\approx}_? (a\ b) \cdot X\}) \qquad \text{(Fixed-point problem)}
\end{array}
$$

Observe that the first step uses the rule $(\approx_{\alpha,\mathsf{C}}\ \text{C})$, which yields two branches, but here, we are interested in analysing only one branch.

The fixed-point problem has infinite solutions, for example:

- $(\{a\#X, b\#X\}, \rho_1)$: instances ρ_1 of X that do not contain free occurrences of a or b. E.g. for $\rho_1 = [X \mapsto g(e)]$, we have $X\rho_1 = g(e) \approx_{\alpha,\mathsf{C}} g(e) = (a\ b) \cdot (X\rho_1)$.
- $(\emptyset, \rho_2 = [X \mapsto a \oplus b])$: since $X\rho_2 = a \oplus b \approx_{\alpha,\mathsf{C}} b \oplus a = (a\ b) \cdot X\rho_2$
- $(\emptyset, \rho_3 = [X \mapsto (a \oplus b) \oplus (a \oplus b)])$: since $X\rho_3 = (a \oplus b) \oplus (a \oplus b) \approx_{\alpha,\mathsf{C}} (b \oplus a) \oplus (b \oplus a) = (a\ b) \cdot X\rho_3$.

3 Nominal E-rewriting and E-narrowing.

In this section, we introduce our novel definitions of *equational nominal rewriting systems* (ENRS) and *nominal equational narrowing*, sometimes abbreviated to nominal E-rewriting systems and nominal E-narrowing.

3.1 Nominal E-rewriting

An *equational nominal rewrite system* (ENRS) is a set of (nominal) identities T that can be split into a set R of nominal rewrite rules and a set E of identities. Sometimes, we will denote this decomposition as R∪E.

Definition 3.1 (Nominal R/E-rewriting)**.** Let $\mathsf{T} = \mathsf{R} \cup \mathsf{E}$ be an ENRS. The relation $\to_{\mathsf{R/E}}$ is induced by the composition $\approx_{\alpha,\mathsf{E}} \circ \to_{\mathsf{R}} \circ \approx_{\alpha,\mathsf{E}}$. A nominal term-in-context $\Delta \vdash s$ reduces with $\to_{\mathsf{R/E}}$, when a term in its E-equivalence class reduces via $\to_{\mathsf{R}}$ as below:

$$\Delta \vdash (s \to_{\mathsf{R/E}} t) \text{ iff there exist } s', t' \text{ such that } \Delta \vdash (s \approx_{\alpha,\mathsf{E}} s' \to_{\mathsf{R}} t' \approx_{\alpha,\mathsf{E}} t).$$

If $\Delta \vdash s \to^*_{\mathsf{R/E}} t$ and $\Delta \vdash s \to^*_{\mathsf{R/E}} u$, then we say that R is E-*confluent* when there exist terms t', u' such that $\Delta \vdash t \to^*_{\mathsf{R/E}} t'$, $\Delta \vdash u \to^*_{\mathsf{R/E}} u'$ and $\Delta \vdash t' \approx_{\alpha,\mathsf{E}} u'$. Also, R is said to be E-*terminating* if there is no infinite $\to_{\mathsf{R/E}}$ sequence. R is called E-*convergent* if it is E-confluent and E-terminating.

The following example illustrates an ENRS for the set of identities that define the prenex normal form of a first-order formula. We consider the commutativity of the connectives $\wedge$ and $\vee$.

Example 3.1 (Prenex normal form rules)**.** Consider the signature for the first-order logic $\Sigma = \{\forall, \exists, \neg, \wedge, \vee\}$, let $\mathsf{C} = \{ \vdash P \vee Q \approx Q \vee P, \vdash P \wedge Q \approx Q \wedge P\}$ be the commutative theory. The prenex normal form rules can be specified by the following set R of nominal rewrite rules:

$$\begin{array}{ll} a\#P & \vdash P \wedge \forall[a]Q \to \forall[a](P \wedge Q) \\ a\#P & \vdash P \vee \forall[a]Q \to \forall[a](P \vee Q) \\ a\#P & \vdash P \wedge \exists[a]Q \to \exists[a](P \wedge Q) \\ a\#P & \vdash P \vee \exists[a]Q \to \exists[a](P \vee Q) \\ & \vdash \neg(\exists[a]Q) \to \forall[a]\neg Q \\ & \vdash \neg(\forall[a]Q) \to \exists[a]\neg Q \end{array}$$

Note that in Definition 3.1, the relation $\to_{\mathsf{R/E}}$ deals with α,E-congruence classes and they are always infinite due to the availability of names for α-renaming. Although the pure $\approx_\alpha$ relation is decidable, when $\approx_\alpha$ is put together with an equational theory E which contains infinite congruence classes, the relation $\to_{\mathsf{R/E}}$ may not be decidable (as in standard first-order rewriting modulo E). We will define the nominal relation $\to_{\mathsf{R,E}}$ that deals with nominal E-matching instead of inspecting the whole α,E-congruence class of a term.

Definition 3.2 (Nominal R,E-rewriting)**.** The *one-step* E-*rewrite relation* $\Delta \vdash s \to_{\mathsf{R,E}} t$ is the least relation such that for any $R = (\nabla \vdash l \to r) \in \mathsf{R}$, position $\mathbb{C}$, term s', permutation π, and substitution θ,

$$\frac{s \equiv \mathbb{C}[s'] \qquad \Delta \vdash \left(\nabla\theta,\ s' \approx_{\alpha,\mathsf{E}} \pi \cdot (l\theta),\ \mathbb{C}[\pi \cdot (r\theta)] \approx_\alpha t\right)}{\Delta \vdash s \to_{\mathsf{R,E}} t}$$

The E-*rewrite relation* $\Delta \vdash s \to^*_{\mathsf{R,E}} t$ is the least relation that includes $\to_{\mathsf{R,E}}$ and is closed by reflexivity and transitivity of $\to_{\mathsf{R,E}}$, i.e., it satisfies:

1. for all Δ, s, s' we have $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} s'$ if $\Delta \vdash s \approx_\alpha s'$;
2. for all Δ, s, t, u we have that $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} t$ and $\Delta \vdash t \to^*_{\mathsf{R},\mathsf{E}} u$ implies $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} u$.

If $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} t$ and $\Delta \vdash s \to^*_{\mathsf{R},\mathsf{E}} u$, then we say that R,E *is* E*-confluent* when there exist terms t', u' such that $\Delta \vdash t \to^*_{\mathsf{R},\mathsf{E}} t'$, $\Delta \vdash u \to^*_{\mathsf{R},\mathsf{E}} u'$ and $\Delta \vdash t' \approx_{\alpha,\mathsf{E}} u'$.

A term t is said to be in R,E*-normal* form (R/E-normal form) whenever one cannot apply another step of $\to_{\mathsf{R},\mathsf{E}}$ ($\to_{\mathsf{R}/\mathsf{E}}$).

Example 3.2 (Cont. Example 3.1)**.** This example illustrates the one-step C-rewrite:

$$a\#P' \vdash S' \vee (\exists[a]Q' \vee P') \to_{\mathsf{R},\mathsf{C}} S' \vee (\exists[a](P' \vee Q'))$$

with the rule $a\#P \vdash P \vee \exists[a]Q \to \exists[a](P \vee Q)$. In fact,

- $\Delta = \{a\#P'\}$ and $\nabla = \{a\#P\}$;
- $s = S' \vee (\exists[a]Q' \vee P') \equiv \mathbb{C}[\exists[a]Q' \vee P'] \equiv \mathbb{C}[s']$;

If we fix $\pi = id$ and $\theta = [P \mapsto P', Q \mapsto Q']$ we have:

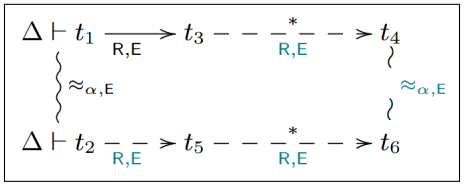
- $\Delta = a\#P' \vdash a\#P' = (a\#P)[P \mapsto P', Q \mapsto Q'] = \nabla\theta$;
- $s' = \exists[a]Q' \vee P' \approx_{\alpha,\mathsf{C}} (P \vee \exists[a]Q)[P \mapsto P', Q \mapsto Q'] = l\theta = \pi \cdot (l\theta)$;
- $\mathbb{C}[\pi \cdot (r\theta)] = \mathbb{C}[r\theta] = \mathbb{C}[(\exists[a](P \vee Q))[P \mapsto P', Q \mapsto Q']] = \mathbb{C}[\exists[a](P' \vee Q')] = S' \vee (\exists[a](P' \vee Q')) \approx_\alpha t$

Thus, $a\#P' \vdash S' \vee (\exists[a]Q' \vee P') \to_{\mathsf{R},\mathsf{C}} S' \vee (\exists[a](P' \vee Q'))$.

Since $\vee$ is a commutative symbol, we could reduce the initial term to three other possible terms because we have two occurrences of the disjunction. Thus, we can "permute" the subterms inside the rewriting modulo C.

Remark 3.3. Following the approach by Jouannaud et al. [18], E-confluence is a consequence of relating $\to_{\mathsf{R}/\mathsf{E}}$ and $\to_{\mathsf{R},\mathsf{E}}$, which relies on a property called E*-coherence* which will be extended here, to the nominal framework.

Definition 3.4 (Nominal E-Coherence)**.** The relation $\Delta \vdash \_ \to_{\mathsf{R},\mathsf{E}} \_$ is called E*-coherent* iff for all t_1, t_2, t_3 such that $\Delta \vdash t_1 \approx_{\alpha,\mathsf{E}} t_2$ and $\Delta \vdash t_1 \to_{\mathsf{R},\mathsf{E}} t_3$, there exist t_4, t_5, t_6 such that $\Delta \vdash t_3 \to^*_{\mathsf{R},\mathsf{E}} t_4$, $t_2 \to_{\mathsf{R},\mathsf{E}} t_5 \to^*_{\mathsf{R},\mathsf{E}} t_6$ and $\Delta \vdash t_4 \approx_{\alpha,\mathsf{E}} t_6$, for some Δ.



The diagram above illustrates nominal E-coherence: the dashed lines represent existentially quantified reductions.

Definition 3.5. An equational theory E is called *a first-order equational theory* iff E is defined via a set of first-order axioms, i.e., identities of the form $\emptyset \vdash l = r$, where l, r are first-order terms. First-order terms do not contain atoms, abstractions and suspended permutations on variables.

Theorem 3.6. *Let* E *be a first-order theory,* R *be a nominal rewrite system that is* E*-terminating and* R,E *be* E*-confluent. Then the* R,E*- and* R/E*-normal forms of any term* t *are* E*-equal iff* $\to_{\mathsf{R},\mathsf{E}}$ *is* E*-coherent.*

In first-order rewriting, it is known that R,E-reducibility is decidable if E-matching is decidable. Following Jouannaud et al. [18], the existence of a finite and complete E-unification algorithm is a sufficient condition for that decidability. However, solving nominal E-unification problems has the additional complication of dealing with α-equality, which significantly impacts obtaining finite and complete sets of nominal E-unifiers.

Remark 3.7. Nominal C-unification is not finitary when one uses freshness constraints and substitutions for representing solutions [2], but the type of problems that generate an infinite set of C-unifiers are fixed-point equations $\pi \cdot X_? \overset{\mathsf{C}}{\approx}_? X$. For example, the nominal C-unification problem $(a\ b)\cdot X_? \overset{\mathsf{C}}{\approx}_? X$ has solutions $[X \mapsto a \oplus b], [X \mapsto (a \oplus b) \oplus (a \oplus b)], \ldots$ (Example 2.2). However, these problems do not appear in nominal C-matching, which is finitary [3]. Thus, the relation $\to_{\mathsf{R},\mathsf{C}}$ is decidable.

3.2 Nominal E-narrowing

Now we define the nominal narrowing relation modulo E, extending previous works [6].

Definition 3.8 (Nominal E-narrowing)**.** The *one-step* E*-narrowing relation* $(\Delta \vdash s) \rightsquigarrow_{\mathsf{R},\mathsf{E}} (\Delta' \vdash t)$ is the least relation such that for any $(\nabla \vdash l \to r) \in \mathsf{R}$, position $\mathbb{C}$, term s', permutation π, and substitution θ,

$$\frac{s \equiv \mathbb{C}[s'] \qquad \Delta' \vdash \left(\nabla\theta,\ \Delta\theta,\ s'\theta \approx_{\alpha,\mathsf{E}} \pi\cdot(l\theta),\ (\mathbb{C}[\pi\cdot r])\theta \approx_\alpha t\right)}{(\Delta \vdash s) \rightsquigarrow^{\theta}_{\mathsf{R},\mathsf{E}} (\Delta' \vdash t)}\,.$$

where $(\Delta', \theta) \in \mathscr{U}_{\mathsf{E}}(\nabla \vdash l, \Delta \vdash s')$. We will write only $(\Delta \vdash s) \rightsquigarrow_{\mathsf{R},\mathsf{E}} (\Delta' \vdash t)$, omitting the θ, when it is clear in the context.

The *nominal* E*-narrowing relation* $(\Delta \vdash s) \rightsquigarrow^*_{\mathsf{R},\mathsf{E}} (\Delta' \vdash t)$ is the least relation that includes $\rightsquigarrow_{\mathsf{R},\mathsf{E}}$ and is closed by reflexivity and transitivity of $\rightsquigarrow_{\mathsf{R},\mathsf{E}}$, i.e., it satisfies:

1. for all Δ, s, s' we have $(\Delta \vdash s) \rightsquigarrow^*_{\mathsf{R},\mathsf{E}} (\Delta \vdash s')$ if $\Delta \vdash s \approx_\alpha s'$;
2. for all $\Delta, \Delta', \Delta'', s, t$ and u: if $(\Delta \vdash s) \rightsquigarrow^*_{\mathsf{R},\mathsf{E}} (\Delta' \vdash t)$ and $(\Delta' \vdash t) \rightsquigarrow^*_{\mathsf{R},\mathsf{E}} (\Delta'' \vdash u)$ then $(\Delta \vdash s) \rightsquigarrow^*_{\mathsf{R},\mathsf{E}} (\Delta'' \vdash u)$.

The permutation π and substitution θ in the definition above are found by solving the nominal E-unification problem $(\nabla \vdash l) \overset{\mathsf{E}}{\approx}_? (\Delta \vdash s')$.

Remark 3.9. Note that decidability of $\rightsquigarrow_{\mathsf{R},\mathsf{E}}$ relies on the existence of an algorithm for nominal E-unification. In this work, we will focus on the theory C, for which a nominal unification algorithm exists.

Since nominal C-narrowing uses nominal C-unification, which is not finitary when we use pairs (Δ', θ) of freshness contexts and substitutions to represent solutions, following Remark 3.7, we conclude that our nominal C-narrowing trees are infinitely branching. The following example illustrates these infinite branches.

Example 3.3 (Cont. Example 2.2)**.** Consider the signature $\Sigma = \{h : 1, f^{\mathsf{C}} : 2, \oplus : 2\}$, where f^{C} and $\oplus$ are commutative symbols. Let $R = \{\ \vdash h(Y) \to Y,\ \vdash f^{\mathsf{C}}([a][b]\cdot Z, Z) \to f^{\mathsf{C}}(h(Z), h(Z))\}$ be a set of rewrite[2] rules. Let $\vdash h(f^{\mathsf{C}}([b][a]X, X))$ be a nominal term that we want to apply nominal C-narrowing to. Observe that we can apply one step of narrowing, and then we obtain a branch that yields infinite branches due to the fixed-point equation (see Figure 3).

[2] $\vdash l \to r$ denotes $\emptyset \vdash l \to r$.

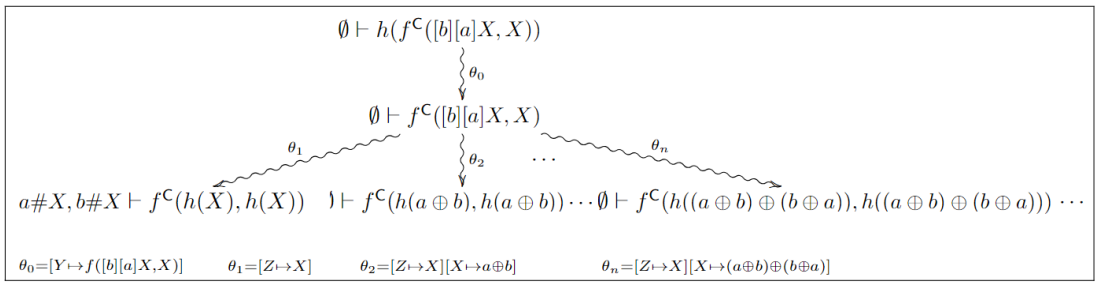


Figure 3: Infinitely branching tree

The first narrowing step is $\emptyset \vdash h(f^{\mathsf{C}}([b][a]X,X)) \rightsquigarrow_{\mathsf{R},\mathsf{C}} \emptyset \vdash f^{\mathsf{C}}([b][a]X,X)$, using the rule $\vdash h(Y) \to Y$. The substitution $\theta_0 = [Y \mapsto f^{\mathsf{C}}([b][a]X,X)]$ was computed in Example 2.2 when solving the C-unification problem $(\emptyset \vdash h(Y)) \stackrel{\mathsf{C}}{\approx}_? (\emptyset \vdash h(f^{\mathsf{C}}([b][a]X,X)))$.

The other infinite narrowing steps are generated due to the fixed-point equation found in the process of solving the C-unification problem $(\emptyset \vdash f^{\mathsf{C}}([a][b]Z,Z)) \stackrel{\mathsf{C}}{\approx}_? (\emptyset \vdash f^{\mathsf{C}}([b][a]X,X))$, computed in Example 2.2. Composing the fixed-point solutions with $(\emptyset, \theta_1)$ that we had, we get the substitutions $\theta_1 = [Z \mapsto X]$, $\theta_2 = [Z \mapsto X][X \mapsto a \oplus b]$ and $\theta_3 = [Z \mapsto X][X \mapsto (a \oplus b) \oplus (a \oplus b)]$ of our narrowing steps in Figure 3.

The following proposition shows that each nominal narrowing step corresponds to a nominal rewriting step, using the same substitution θ.

Proposition 3.10. *Let* E *be an equational theory for which a complete* E*-unification algorithm exists.* $(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta}_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1)$ *implies* $\Delta_1 \vdash (s_0\theta) \to_{\mathsf{R},\mathsf{E}} s_1$.

Proof. Indeed, suppose we have $(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta}_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1)$. The narrowing step guarantees that for a substitution θ, some permutation π, and a rule $\nabla \vdash l \to r \in \mathsf{R}$, the following holds:

- $s_0 \equiv \mathbb{C}[s'_0]$ and $\Delta_1 \vdash \big(\nabla\theta,\ \Delta_0\theta,\ s'_0\theta \approx_{\alpha,\mathsf{E}} \pi \cdot (l\theta),\ (\mathbb{C}[\pi \cdot r])\theta \approx_{\alpha} s_1\big)$.

From the items above, it is easy to verify the following:

- $s_0\theta \equiv \mathbb{C}\theta[s'_0\theta]$; and $\Delta_1 \vdash (\nabla\theta', s'_0\theta \approx_{\alpha,\mathsf{E}} \pi \cdot (l\theta'), \mathbb{C}\theta[\pi \cdot (r\theta')] \approx_{\alpha} s_1)$,

and by the definition of rewrite modulo E, it implies that $\Delta_1 \vdash s_0\theta \to_{\mathsf{R},\mathsf{E}} s_1$. We need to fix the substitution θ used in the narrowing step as θ', and the result follows. □

4 Nominal Lifting Theorem modulo E

In this section, we assume $\mathsf{R} \cup \mathsf{E}$ an ENRS such that $\mathsf{R} = \{\nabla_i \vdash l_i \to r_i\}$ is E-convergent NRS, E is compatible with $\vdash$ and substitutions and that there exists a complete E-unification algorithm. We want to extend Proposition 3.10 and establish correspondence between finite sequences of nominal E-narrowing steps and sequences of nominal E-rewriting steps. This result corresponds to the classical Lifting Theorem ([16, 18, 6]) which will be extended to the nominal relations $\rightsquigarrow_{\mathsf{R},\mathsf{E}}$ and $\to_{\mathsf{R},\mathsf{E}}$. The Lifting Theorem relates narrowing steps to rewriting steps. It is fundamental to guarantee that one can use the narrowing relation to solve T-unification problems when T is a convergent equational theory. The extension to the $\mathsf{R} \cup \mathsf{E}$-Lifting Theorem would allow us to solve nominal unification problems modulo $\mathsf{R} \cup \mathsf{E}$.

We start by defining a normalised substitution with respect to the relation $\to_{\mathsf{R},\mathsf{E}}$:

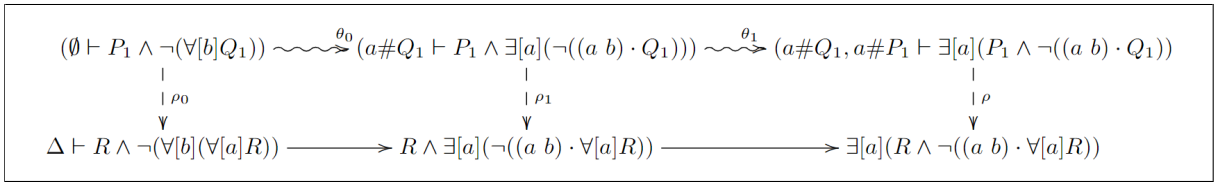


Figure 4: Illustration of Example 4.1

Definition 4.1 (Normalised substitution w.r.t $\to_{\mathsf{R},\mathsf{E}}$)**.** A substitution θ is *normalised in* Δ *with relation to* $\to_{\mathsf{R},\mathsf{E}}$ if $\Delta \vdash X\theta$ is a R,E-normal form for every X. A substitution θ *satisfies the freshness context* Δ iff there exists a freshness context ∇ such that $\nabla \vdash a\#X\theta$ for each $a\#X \in \Delta$. In this case, we say that θ satisfies Δ with ∇. The minimal such ∇ is $\langle \Delta\theta \rangle_{nf}$.

The following example illustrates the technique used in the proof of Lemma 4.2.

Example 4.1. Consider the rules $R_3 : a\#P \vdash P \wedge \exists[a]Q \to \exists[a](P \wedge Q)$ and $R_6 : \emptyset \vdash \neg(\forall[a]Q) \to \exists[a]\neg Q$.

Let $(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta_0}_{R_6} (\Delta_1 \vdash s_1) \rightsquigarrow^{\theta_1}_{R_3} (\Delta_2 \vdash s_2)$ be a narrowing derivation, illustrated in Figure 4 such that:

- $\Delta_0 \equiv \emptyset$ and $s_0 \equiv P_1 \wedge \neg(\forall[b]Q_1)$
- $\Delta_1 \equiv \{a\#Q_1\}$ and $s_1 \equiv P_1 \wedge \exists[a](\neg(a\ b)\cdot Q_1)$
- $\Delta_2 \equiv \{a\#Q_1, a\#P_1\}$ and $s_2 \equiv \exists[a](P_1 \wedge \neg(a\ b)\cdot Q_1)$

Let ρ be a substitution that satisfies Δ_2 with Δ. Then, there exists a rewriting derivation

$$\Delta \vdash s_0\rho_0 \to_{\mathsf{R},\mathsf{C}} s_1\rho_1 \to_{\mathsf{R},\mathsf{C}} s_2\rho$$

where $\Delta \vdash \Delta_0\rho_0$, $\Delta \vdash \Delta_1\rho_1$ and $\rho_0 = \theta_0\theta_1\rho$, $\rho_1 = \theta_1\rho$.

Supposing that $\rho = [Q_1 \mapsto \forall[a]R, P_1 \mapsto R]$, and $\Delta = \{a\#R\}$, we have

- $\Delta \vdash \Delta_2\rho = \{a\#Q_1, a\#P_1\}\rho = \{a\#\forall[a]R, a\#R\} = \{a\#R\}$
- $\theta_0 = [Q \mapsto (a\ b)\cdot Q_1]$ and $\theta_1 = [P' \mapsto P_1, Q' \mapsto \neg((a\ b)\cdot Q_1)]$
- $\rho_1 = \theta_1\rho = [P' \mapsto R, Q' \mapsto \neg((a\ b)\cdot\forall[a]R), Q_1 \mapsto \forall[a]R, P_1 \mapsto R]$
- $\rho_0 = \theta_0\rho_1 = [Q \mapsto (a\ b)\cdot\forall[a]R, P' \mapsto R, Q' \mapsto \neg((a\ b)\cdot\forall[a]R), Q_1 \mapsto \forall[a]R, P_1 \mapsto R]$
- $\Delta \vdash \Delta_1\rho_1 = (a\#Q_1)\rho_1 = a\#\forall[a]R = \emptyset$ and $\Delta \vdash \Delta_0\rho_0 = \emptyset$

The next result shows that the rewriting step generated by the narrowing step is preserved by application of substitution if the theory E is compatible (Definition 2.3), that is, $\vdash$ is closed by substitutions.

Lemma 4.2. *($\rightsquigarrow_{\mathsf{R},\mathsf{E}}$ to $\to_{\mathsf{R},\mathsf{E}}$) Let* E *be compatible with* $\vdash$ *by substitutions and* $(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta}_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1)$. *Then, for any substitution* ρ *that satisfies* Δ_1 *with* Δ*, the following holds*

$$\Delta \vdash (s_0\theta)\rho \to_{\mathsf{R},\mathsf{E}} s_1\rho$$

In particular, Δ *will be* $\langle \Delta_1\rho \rangle_{nf}$.

Proof. From Proposition 3.10: $(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta}_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1)$ implies $\Delta_1 \vdash (s_0\theta) \to_{\mathsf{R},\mathsf{E}} s_1$. By E-compatiblity of derivability with substitutions $\Delta_1 \vdash s_0\theta \to_{\mathsf{R},\mathsf{E}} s_1$ gives:

- $(s_0\theta)\rho \equiv (\mathbb{C}\theta[s'_0\theta])\rho = \mathbb{C}\theta\rho[(s'_0\theta)\rho]$

- $\Delta_1 \vdash \nabla\theta$ implies $\langle\Delta_1\rho\rangle_{nf} \vdash \nabla\theta\rho$
- $\Delta_1 \vdash s_0'\theta \approx_{\alpha,\mathsf{E}} \pi\cdot(l\theta)$ implies $\langle\Delta_1\rho\rangle_{nf} \vdash s_0'\theta\rho \approx_{\alpha,\mathsf{E}} (\pi\cdot(l\theta))\rho = \pi\cdot(l\theta\rho)$
- $\Delta_1 \vdash \mathbb{C}\theta[\pi\cdot(r\theta)] \approx_\alpha s_1$ implies $\langle\Delta_1\rho\rangle_{nf} \vdash \mathbb{C}\theta\rho[\pi\cdot(r\theta\rho)] = (\mathbb{C}\theta[\pi\cdot(r\theta)])\rho \approx_\alpha s_1\rho$

which implies that $\langle\Delta_1\rho\rangle_{nf} \vdash (s_0\theta)\rho \to_{\mathsf{R},\mathsf{E}} s_1\rho$. Note that we need ρ satisfying Δ_1 with Δ to guarantee that when we instantiate Δ_1 we do not have any inconsistency with the freshness constraints in Δ_1. □

The following result (correctness) states that a finite sequence of rewriting steps exists for each finite sequence of narrowing steps.

Lemma 4.3. *($\rightsquigarrow^*_{\mathsf{R},\mathsf{E}}$ to $\to^*_{\mathsf{R},\mathsf{E}}$) Let E be compatible with $\vdash$ by substitutions and $(\Delta_0 \vdash s_0) \rightsquigarrow^*_{\mathsf{R},\mathsf{E}} (\Delta_n \vdash s_n)$ be a nominal E-narrowing derivation. Let ρ be a substitution satisfying Δ_n with Δ.*

$$(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta_0}_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1) \rightsquigarrow^{\theta_1}_{\mathsf{R},\mathsf{E}} \dots \rightsquigarrow^{\theta_{n-1}}_{\mathsf{R},\mathsf{E}} (\Delta_n \vdash s_n)$$

Then, there exists a nominal E-rewriting derivation

$$\Delta \vdash s_0\rho_0 \to_{\mathsf{R},\mathsf{E}} \dots \to_{\mathsf{R},\mathsf{E}} s_i\rho_i \to_{\mathsf{R},\mathsf{E}} \dots \to_{\mathsf{R},\mathsf{E}} s_{n-1}\rho_{n-1} \to_{\mathsf{R},\mathsf{E}} s_n\rho$$

such that $\Delta \vdash \Delta_i\rho_i$ and $\rho_i = \theta_i\dots\theta_{n-1}\rho$, for all $0 \le i < n$. In other words, $\Delta \vdash (s_0\theta)\rho \to^*_{\mathsf{R},\mathsf{E}} s_n\rho$ where $\theta = \theta_0\theta_1\dots\theta_{n-1}$.

Proof. By induction on the length $n \ge 1$ of the narrowing derivation $(\Delta_0 \vdash s_0) \rightsquigarrow^n_{\mathsf{R},\mathsf{E}} (\Delta_n \vdash s_n)$, using the one-step result proved in Lemma 4.2. (We start the induction for $n = 1$ because the case for $n = 0$ holds trivially and gives no additional insight.)

- **Base Case:** For $n = 1$, we have $(\Delta_0 \vdash s_0) \rightsquigarrow_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1)$ and by Lemma 4.2, for any ρ satisfying Δ_1 with Δ we have $\Delta \vdash (s_0\theta_0)\rho \to_{\mathsf{R},\mathsf{E}} s_1\rho$. Since $\Delta \vdash \Delta_1\rho$, and by the narrowing step $\Delta_1 \vdash \Delta_0\theta_0$, we get $\Delta \vdash \Delta_0\theta_0\rho$. Taking $\rho_0 = \theta_0\rho$, we have the result $\Delta \vdash s_0\rho_0 \to_{\mathsf{R},\mathsf{E}} s_1\rho$ such that $\Delta \vdash \Delta_0\rho_0$.
- **Induction Step:** Assume that the result holds for $n > 1$. Then $(\Delta_0 \vdash s_0) \rightsquigarrow^n_{\mathsf{R},\mathsf{E}} (\Delta_n \vdash s_n)$ implies that there exists a rewriting derivation $\Delta \vdash s_0\rho_0 \to^n_{\mathsf{R},\mathsf{E}} s_n\rho$, for some ρ satisfying Δ_n with Δ and Figure 5 illustrates this setting.

  We want to show that the result follows for $n+1$. Consider the narrowing step

  $$(\Delta_n \vdash s_n) \rightsquigarrow^{\theta_n}_{\mathsf{R},\mathsf{E}} (\Delta_{n+1} \vdash s_{n+1}).$$

  By Lemma 4.2, for any substitution, let's name it σ, that satisfies Δ_{n+1} with Δ **(H1)** we have

  $$\Delta \vdash (s_n\theta_n)\sigma \to_{\mathsf{R},\mathsf{E}} s_{n+1}\sigma \tag{1}$$

  Take $\rho = \theta_n\sigma$. Note that ρ satisfies Δ_n with Δ:

  (H1) $\Delta \vdash \Delta_{n+1}\sigma$.

  (H2) By Definition 3.8: $\Delta_{n+1} \vdash \Delta_n\theta_n$.

  (H3) From **(H2)** and E-compatibility Definition 2.3(1) generalised to E: $\langle\Delta_{n+1}\sigma\rangle_{nf} \vdash \Delta_n\rho$.

  Thus, from **(H1)** and **(H3)** it follows that $\Delta \vdash \Delta_n\rho$. By the induction hypothesis, we have

  $$\Delta \vdash s_0\theta_0\dots\theta_{n-1}\rho \to^n_{\mathsf{R},\mathsf{E}} s_n\rho$$

  with $\Delta \vdash \Delta_i\rho_i$ and $\rho_i = \theta_i\dots\theta_{n-1}\rho$, for every $i = 1,\dots,n$. Hence,

  $$\Delta \vdash s_0\theta_0\dots\theta_{n-1}\theta_n\sigma \to^n_{\mathsf{R},\mathsf{E}} s_n\theta_n\sigma \overset{(1)}{\to}_{\mathsf{R},\mathsf{E}} s_{n+1}\sigma,$$

  and the result follows.

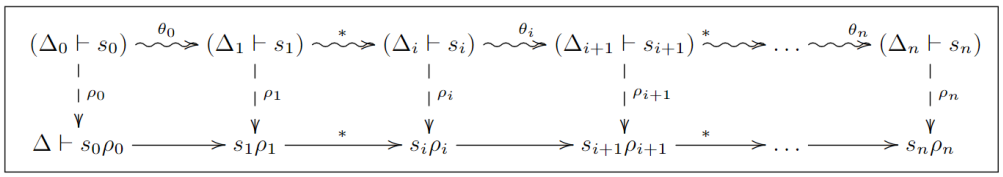


Figure 5: Corresponding Narrowing to Rewriting Derivations

□

The proof of the converse (completeness) is more challenging. Nevertheless, for one-step rewriting to one-step narrowing, the result holds with no further problems:

Lemma 4.4. *($\to_{\mathsf{R},\mathsf{E}}$ to $\rightsquigarrow_{\mathsf{R},\mathsf{E}}$) Let $\Delta_0 \vdash s_0$ be a nominal term in context and V_0 a finite set of variables containing $V = V(\Delta_0, s_0)$. Let ρ_0 be a R,E-normalised substitution, with $\mathtt{dom}(\rho_0) \subseteq V$, that satisfies Δ_0 with Δ and*

$$\Delta \vdash s_0\rho_0 = t_0 \to_{\mathsf{R},\mathsf{E}} t_1.$$

Then, there exists a nominal R,E-narrowing step

$$(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta}_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1),$$

for a substitution θ, a finite set of variables $V_1 \supseteq V(s_0)$, and a R,E-normalised substitution ρ_1 with Δ such that

$$(i)\ \Delta \vdash s_1\rho_1 \approx_{\alpha,\mathsf{E}} t_1 \qquad (ii)\ \mathtt{dom}(\rho_1) \subseteq V_1 \qquad (iii)\ \Delta \vdash \rho_0|_V \approx_{\alpha,\mathsf{E}} \theta\rho_1|_V$$

Proof. Suppose that the one-step rewriting is done in a position $\mathbb{C}_0$ of t_0, with substitution σ and rule $R_0 = \nabla_0 \vdash l_0 \to r_0 \in \mathsf{R}$:

$$(*)\ \frac{t_0 \equiv \mathbb{C}_0[t_0'] \qquad \Delta \vdash \nabla_0\sigma,\ t_0' \approx_{\alpha,\mathsf{E}} \pi\cdot(l_0\sigma),\ \mathbb{C}_0[\pi\cdot(r_0\sigma)] \approx_{\alpha} t_1}{\Delta \vdash t_0 \to_{[\mathbb{C}_0,R_0],\mathsf{E}} t_1}$$

The following hold:

(H1) The variables of R_0 are renamed with respect to $t_0 = s_0\rho_0$ and Δ (to avoid conflicts). Thus, $V(R_0) \cap V(\Delta, t_0) = \emptyset$ and $\mathtt{dom}(\sigma) \cap V_0 = \emptyset$.

(H2) By hypothesis, $\Delta \vdash \Delta_0\rho_0$

(H3) Since ρ_0 is normalised in Δ and $\Delta \vdash s_0\rho_0 \to_{\mathsf{R},\mathsf{E}} t_1$, there must exist a non-variable position $\mathbb{C}_0'$ and a subterm s_0' of s_0 such that $s_0 \equiv \mathbb{C}_0'[s_0']$ and $\Delta \vdash s_0'\rho_0 \approx_{\alpha,\mathsf{E}} t_0' \approx_{\alpha,\mathsf{E}} \pi\cdot(l_0\sigma)$.

Define **(H4)** $\theta = \rho_0\sigma$. Then, we have the following:

(H5) $\Delta \vdash \Delta_0\theta$: from **(H2)** it follows that $\Delta \vdash \Delta_0\rho_0$ and σ does not affect Δ_0 since $\mathtt{dom}(\sigma) = V(R_0)$.

(H6) Note that $s_0'\theta = s_0'\rho_0\sigma = s_0'\rho_0$ from **(H1)**. Therefore, $\Delta \vdash s_0'\theta \approx_{\alpha,\mathsf{E}} \pi\cdot(l_0\theta)$ and $\Delta \vdash \nabla_0\theta$, and (Δ, θ) is a solution for the nominal E-unification problem $(\Delta_0 \vdash s_0') \stackrel{?}{\approx}_{\mathsf{E}}^{?} (\nabla_0 \vdash \pi\cdot l_0)$. That is, $(\Delta, \theta) \in \mathscr{U}_{\mathsf{E}}(\Delta_0 \vdash s_0', \nabla_0 \vdash \pi\cdot l_0)$.

Define s_1 as $s_1 = \mathbb{C}'_0[\pi \cdot r_0]\theta$ and $\Delta_1 = \Delta$. Conditions **(H4)** to **(H6)** imply the existence of the following nominal E-narrowing step:

$$(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta}_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1).$$

Also $\Delta \vdash s_1 = \mathbb{C}'_0[\pi \cdot r_0]\theta = (\mathbb{C}'_0[\pi \cdot r_0])\rho_0\sigma \approx_\alpha (\mathbb{C}'_0\rho_0)[\pi \cdot r_0]\rho_0 \overset{(*)}{\approx}_\alpha t_1$. Take $\rho_1 = \mathtt{Id}$ as the identity substitution, and items (i) and (ii) follow with $\mathtt{dom}(\rho_1) = \emptyset$ and $V_1 = V(s_0)$. Finally, it is trivial to check (iii): $\Delta \vdash X\rho_0 \approx_{\alpha,\mathsf{E}} X\theta\rho_1 \equiv X\rho_0\sigma\mathtt{Id}$, for all $X \in V$, and the result follows.

□

Lemma 4.5. *($\rightarrow^*_{\mathsf{R},\mathsf{E}}$ to $\rightsquigarrow^*_{\mathsf{R},\mathsf{E}}$) Let $\mathsf{R} \cup \mathsf{E}$ be an ENRS such that R is E-terminating, R,E is E-confluent and $\rightarrow_{\mathsf{R},\mathsf{E}}$ is E-coherent. Let V_0 be a finite set of variables containing $V = V(\Delta_0, s_0)$. Then, for any R,E-derivation*

$$\Delta \vdash t_0 = s_0\rho_0 \rightarrow_{\mathsf{R},\mathsf{E}} t_1 \rightarrow_{\mathsf{R},\mathsf{E}} \ldots \rightarrow_{\mathsf{R},\mathsf{E}} t_n = t_0{\downarrow}$$

to any of its R,E-normal forms, say $t_0{\downarrow}$, where $\mathtt{dom}(\rho_0) \subseteq V(s_0) \subseteq V_0$ and ρ_0 is a R,E-normalised substitution that satisfies Δ_0 with Δ, there exist a R,E-narrowing derivation

$$(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta_0}_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1) \rightsquigarrow^{\theta_1}_{\mathsf{R},\mathsf{E}} \ldots \rightsquigarrow^{\theta_{n-1}}_{\mathsf{R},\mathsf{E}} (\Delta_n \vdash s_n)$$

$$\textit{(1)}\ \Delta \vdash \Delta_i\rho_i; \qquad \textit{(2)}\ \Delta \vdash s_i\rho_i \approx_{\alpha,\mathsf{E}} t_i; \qquad \textit{(3)}\ \Delta \vdash \rho_0|_V \approx_{\alpha,\mathsf{E}} \theta\rho_n|_V.$$

where $\rho_i = \theta_i \ldots \theta_{n-1}\rho$ and $\theta = \theta_0\theta_1 \ldots \theta_{n-1}$.

Proof. By induction on the number of steps n applied in the derivation $\Delta \vdash t_0 = s_0\rho_0 \rightarrow^*_{\mathsf{R},\mathsf{E}} t_0{\downarrow}$.

- **Base Case:** For $n = 1$ the result follows directly from Lemma 4.4.
- **Induction Step:** Let $n > 1$ and assume that the result holds for sequences of $n-1$ rewriting steps. Then,

  $$\Delta \vdash t_0 = s_0\rho_0 \rightarrow_{\mathsf{R},\mathsf{E}} t_1 \overbrace{\rightarrow_{\mathsf{R},\mathsf{E}} \ldots \rightarrow_{\mathsf{R},\mathsf{E}}}^{n-1} t_n = t_0{\downarrow}$$

  Now using Lemma 4.4 on the rewrite step $\Delta \vdash t_0 \rightarrow_{\mathsf{R},\mathsf{E}} t_1$. Then, we get that $(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta_0}_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1)$, where ρ_0 is a R,E-normalised substitution that satisfies Δ_0 with Δ, and

  (H1) $\Delta \vdash s_1\rho_1 = t'_1 \approx_{\alpha,\mathsf{E}} t_1$; and

  (H2) $\Delta \vdash \rho_0|_V \approx_{\alpha,\mathsf{E}} \theta\rho_1|_V$.

  Now consider the sequence to any of the normal forms of t_1:

  $$\Delta \vdash t_1 \overbrace{\rightarrow_{\mathsf{R},\mathsf{E}} \ldots \rightarrow_{\mathsf{R},\mathsf{E}}}^{n-1} t_n = t_1{\downarrow}_{\mathsf{R},\mathsf{E}}$$

  By the induction hypothesis, there exists a narrowing sequence

  $$(\Delta_1 \vdash s_1) \rightsquigarrow^{\theta_1}_{\mathsf{R},\mathsf{E}} \ldots \rightsquigarrow^{\theta_{n-1}}_{\mathsf{R},\mathsf{E}} (\Delta_n \vdash s_n)$$

  with $\theta = \theta_1 \ldots \theta_{n-1}$, a normalised substitution ρ_n such that

  (H3) $\Delta \vdash \Delta_i\rho_i$;

  (H4) $\Delta \vdash s_i\rho_i = t'_i \approx_{\alpha,\mathsf{E}} t_i$, for every i;

  (H5) $\Delta \vdash \rho_0|_V \approx_{\alpha,\mathsf{E}} \theta\rho_n|_V$.

Note that from **(H4)**, $\Delta \vdash s_n\rho_n = t'_n \approx_{\alpha,\mathsf{E}} t_n$. Since R is E-convergent and $\to_{\mathsf{R},\mathsf{E}}$ is E-coherent, it follows from Theorem 3.6, that all the normal forms of t_0 are $\approx_{\alpha,\mathsf{E}}$-equivalent. That is, $\Delta \vdash t'_n \approx_{\alpha,\mathsf{E}} t_1 \downarrow_{\mathsf{R},\mathsf{E}} \approx_{\alpha,\mathsf{E}} t_0 \downarrow_{\mathsf{R},\mathsf{E}} = t_n$. Therefore, there exists a nominal E-narrowing sequence

$$(\Delta_0 \vdash s_0) \rightsquigarrow^{\theta_0}_{\mathsf{R},\mathsf{E}} (\Delta_1 \vdash s_1) \rightsquigarrow^{\theta_1}_{\mathsf{R},\mathsf{E}} \dots \rightsquigarrow^{\theta_{n-1}}_{\mathsf{R},\mathsf{E}} (\Delta_n \vdash s_n).$$

□

As a consequence of Lemmas 4.3 and 4.5 we obtain:

Theorem 4.6 (E-Lifting Theorem)**.** *Let* $\mathsf{R} \cup \mathsf{E}$ *be an ENRS such that* E *is compatible with* $\vdash$ *by substitutions,* R *is* E*-terminating,* R,E *is* E*-confluent and* $\to_{\mathsf{R},\mathsf{E}}$ *is* E*-coherent. To each finite sequence of nominal* E*-rewriting steps corresponds a finite sequence of nominal* E*-narrowing steps, and vice versa.*

Since there exists an algorithm for nominal C-unification and C is compatible with substitutions (Proposition 2.4), we have the following result.

Corollary 4.7. *The* C*-Nominal Lifting theorem holds.*

5 Conclusion and Future Work

In this work, we proposed definitions for nominal R,E-rewriting and R,E-narrowing and proved some properties relating them, obtaining the proof of the E-Lifting Theorem, in the case R is an E-convergent NRS, $\to_{\mathsf{R},\mathsf{E}}$ is E-coherent and a complete algorithm for nominal E-unification exists. As C is the only equational theory for which a complete algorithm for nominal unification exists, we illustrate our results using this theory. Also, since the nominal C-unification problem (when using freshness constraints) only is finitary, our nominal C-narrowing tree is infinitely branching. In future work, we plan to investigate alternative approaches to nominal C-unification for which the representation of solutions is finite, such as the approach using fixed-point constraints.

References

[1] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández & Daniele Nantes-Sobrinho (2017): *Nominal C-Unification*. In Fabio Fioravanti & John P. Gallagher, editors: *Logic-Based Program Synthesis and Transformation - 27th International Symposium, LOPSTR 2017, Namur, Belgium, October 10-12, 2017, Revised Selected Papers*, *Lecture Notes in Computer Science* 10855, Springer, pp. 235–251, doi:10.1007/978-3-319-94460-9_14.

[2] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández & Daniele Nantes-Sobrinho (2017): *On Solving Nominal Fixpoint Equations*. In Clare Dixon & Marcelo Finger, editors: *Frontiers of Combining Systems - 11th International Symposium, FroCoS 2017, Brasília, Brazil, September 27-29, 2017, Proceedings*, *Lecture Notes in Computer Science* 10483, Springer, pp. 209–226, doi:10.1007/978-3-319-66167-4_12.

[3] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández & Daniele Nantes-Sobrinho (2018): *A Formalisation of Nominal C-Matching through Unification with Protected Variables*. In Beniamino Accattoli & Carlos Olarte, editors: *Proceedings of the 13th Workshop on Logical and Semantic Frameworks with Applications, LSFA 2018, Fortaleza, Brazil, September 26-28, 2018*, *Electronic Notes in Theoretical Computer Science* 344, Elsevier, pp. 47–65, doi:10.1016/j.entcs.2019.07.004.

[4] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, Daniele Nantes-Sobrinho & Ana Cristina Rocha Oliveira (2019): *A formalisation of nominal α-equivalence with A, C, and AC function symbols*. *Theor. Comput. Sci.* 781, pp. 3–23, doi:10.1016/j.tcs.2019.02.020.

[5] Mauricio Ayala-Rincón, Washington de Carvalho Segundo, Maribel Fernández, Gabriel Ferreira Silva & Daniele Nantes-Sobrinho (2021): *Formalising nominal C-unification generalised with protected variables*. *Math. Struct. Comput. Sci.* 31(3), pp. 286–311, doi:10.1017/S0960129521000050.

[6] Mauricio Ayala-Rincón, Maribel Fernández & Daniele Nantes-Sobrinho (2016): *Nominal Narrowing*. In Delia Kesner & Brigitte Pientka, editors: *1st International Conference on Formal Structures for Computation and Deduction, FSCD 2016, June 22-26, 2016, Porto, Portugal*, *LIPIcs* 52, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 11:1–11:17, doi:10.4230/LIPIcs.FSCD.2016.11.

[7] Mauricio Ayala-Rincón, Maribel Fernández & Daniele Nantes-Sobrinho (2018): *Fixed-Point Constraints for Nominal Equational Unification*. In Hélène Kirchner, editor: *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, *LIPIcs* 108, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 7:1–7:16, doi:10.4230/LIPIcs.FSCD.2018.7.

[8] Mauricio Ayala-Rincón, Maribel Fernández & Ana Cristina Rocha Oliveira (2015): *Completeness in PVS of a Nominal Unification Algorithm*. In Mario R. F. Benevides & René Thiemann, editors: *Proceedings of the Tenth Workshop on Logical and Semantic Frameworks, with Applications, LSFA 2015, Natal, Brazil, August 31 - September 1, 2015*, *Electronic Notes in Theoretical Computer Science* 323, Elsevier, pp. 57–74, doi:10.1016/j.entcs.2016.06.005.

[9] Mauricio Ayala-Rincón, Maribel Fernández, Gabriel Ferreira Silva, Temur Kutsia & Daniele Nantes-Sobrinho (2023): *Nominal AC-Matching*. In Catherine Dubois & Manfred Kerber, editors: *Intelligent Computer Mathematics - 16th International Conference, CICM 2023, Cambridge, UK, September 5-8, 2023, Proceedings*, *Lecture Notes in Computer Science* 14101, Springer, pp. 53–68, doi:10.1007/978-3-031-42753-4_4.

[10] Franz Baader & Tobias Nipkow (1998): *Term rewriting and all that*. Cambridge University Press, doi:10.1017/CBO9781139172752.

[11] Christophe Calvès & Maribel Fernández (2010): *Matching and alpha-equivalence check for nominal terms*. *J. Comput. Syst. Sci.* 76(5), pp. 283–301, doi:10.1016/j.jcss.2009.10.003.

[12] Jesús Domínguez & Maribel Fernández (2019): *Nominal Syntax with Atom Substitutions: Matching, Unification, Rewriting*. In Leszek Antoni Gasieniec, Jesper Jansson & Christos Levcopoulos, editors: *Fundamentals of Computation Theory - 22nd International Symposium, FCT 2019, Copenhagen, Denmark, August 12-14, 2019, Proceedings*, *Lecture Notes in Computer Science* 11651, Springer, pp. 64–79, doi:10.1007/978-3-030-25027-0_5.

[13] Santiago Escobar, José Meseguer & Ralf Sasse (2008): *Variant Narrowing and Equational Unification*. In Grigore Rosu, editor: *Proceedings of the Seventh International Workshop on Rewriting Logic and its Applications, WRLA 2008, Budapest, Hungary, March 29-30, 2008*, *Electronic Notes in Theoretical Computer Science* 238, Elsevier, pp. 103–119, doi:10.1016/j.entcs.2009.05.015.

[14] Maribel Fernández & Murdoch Gabbay (2007): *Nominal rewriting*. *Inf. Comput.* 205(6), pp. 917–965, doi:10.1016/j.ic.2006.12.002.

[15] Murdoch Gabbay & Andrew M. Pitts (2002): *A New Approach to Abstract Syntax with Variable Binding*. *Formal Aspects Comput.* 13(3-5), pp. 341–363, doi:10.1007/s001650200016.

[16] Jean-Marie Hullot (1980): *Canonical Forms and Unification*. In Wolfgang Bibel & Robert A. Kowalski, editors: *5th Conference on Automated Deduction, Les Arcs, France, July 8-11, 1980, Proceedings*, *Lecture Notes in Computer Science* 87, Springer, pp. 318–334, doi:10.1007/3-540-10009-1_25.

[17] Jean-Pierre Jouannaud (1983): *Confluent and Coherent Equational Term Rewriting Systems: Application to Proofs in Abstract Data Types*. In Giorgio Ausiello & Marco Protasi, editors: *CAAP'83, Trees in Algebra and Programming, 8th Colloquium, L'Aquila, Italy, March 9-11, 1983, Proceedings*, *Lecture Notes in Computer Science* 159, Springer, pp. 269–283, doi:10.1007/3-540-12727-5_16.

[18] Jean-Pierre Jouannaud, Claude Kirchner & Hélène Kirchner (1983): *Incremental Construction of Unification Algorithms in Equational Theories*. In Josep Díaz, editor: *Automata, Languages and Programming, 10th*

Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings, *Lecture Notes in Computer Science* 154, Springer, pp. 361–373, doi:10.1007/BFb0036921.

[19] Kentaro Kikuchi (2022): *Ground Confluence and Strong Commutation Modulo Alpha-Equivalence in Nominal Rewriting*. In Helmut Seidl, Zhiming Liu & Corina S. Pasareanu, editors: *Theoretical Aspects of Computing - ICTAC 2022 - 19th International Colloquium, Tbilisi, Georgia, September 27-29, 2022, Proceedings*, *Lecture Notes in Computer Science* 13572, Springer, pp. 255–271, doi:10.1007/978-3-031-17715-6_17.

[20] Kentaro Kikuchi & Takahito Aoto (2020): *Confluence and Commutation for Nominal Rewriting Systems with Atom-Variables*. In Maribel Fernández, editor: *Logic-Based Program Synthesis and Transformation - 30th International Symposium, LOPSTR 2020, Bologna, Italy, September 7-9, 2020, Proceedings*, *Lecture Notes in Computer Science* 12561, Springer, pp. 56–73, doi:10.1007/978-3-030-68446-4_3.

[21] Manfred Schmidt-Schauß, Temur Kutsia, Jordi Levy, Mateu Villaret & Yunus D. K. Kutz (2022): *Nominal Unification and Matching of Higher Order Expressions with Recursive Let*. *Fundam. Informaticae* 185(3), pp. 247–283, doi:10.3233/FI-222110.

[22] Christian Urban, Andrew M. Pitts & Murdoch Gabbay (2004): *Nominal unification*. *Theor. Comput. Sci.* 323(1-3), pp. 473–497, doi:10.1016/j.tcs.2004.06.016.

[23] Emanuele Viola (2001): *E-unifiability via Narrowing*. In Antonio Restivo, Simona Ronchi Della Rocca & Luca Roversi, editors: *Theoretical Computer Science, 7th Italian Conference, ICTCS 2001, Torino, Italy, October 4-6, 2001, Proceedings*, *Lecture Notes in Computer Science* 2202, Springer, pp. 426–438, doi:10.1007/3-540-45446-2_27.

Towards an Analysis of Proofs in Arithmetic

Alexander Leitsch
Institute of Logic and Computation,
TU Wien, Vienna, Austria
leitsch@logic.at

Anela Lolić *
Kurt Gödel Society
Institute of Logic and Computation,
TU Wien, Vienna, Austria
anela@logic.at

Stella Mahler †
Institute of Logic and Computation,
TU Wien, Vienna, Austria
stella@logic.at

Inductive proofs can be represented as proof schemata, i.e. as parameterized sequences of proofs defined in a primitive recursive way. Applications of proof schemata can be found in the area of automated proof analysis where the schemata admit (schematic) cut-elimination and the construction of Herbrand systems. This work focuses on the expressivity of proof schemata. We show that proof schemata can simulate primitive recursive arithmetic. The translation of proofs in arithmetic to proof schemata can be considered as a crucial step in the analysis of inductive proofs.

1 Introduction

Mathematical induction is one of the most important principles in real mathematics, thus any substantial and relevant approach to analyze mathematical proofs has to take into account induction. But in systems with induction rules, essential proof theoretic concepts and transformation become problematic. In particular Gentzen's method of cut-elimination fails for general induction proofs and Herbrand's theorem cannot be realized [12]. The reason is that in the cut-elimination method à la Gentzen cuts cannot be shifted over induction rules. We will illustrate this problem on a concrete example below.

Example 1. *Let $P(x)$ denote $h(x) = 0$ for a primitive recursive function defined in primitive recursive arithmetic* **PRA***, and $g(x,y)$ be any primitive recursive function in* **PRA***. Moreover, let f be a unary function and*

$$\mathscr{E} = \{f(x,0) = x, \quad f(x,s(y)) = s(f(x,y))\}$$

(defining addition). Now we consider the sequent

$$S\colon \forall x(P(x) \to P(s(x))) \vdash \forall n \forall x((P(f(x,n)) \to P(g(x,n))) \to (P(x) \to P(g(x,n)))).$$

S is not valid in pure first-order logic and does not have a Herbrand sequent w.r.t. to the theory $\mathscr{E}$ and hence cannot be proven without induction. Therefore, there is no proof of S in pure first-order logic. We need the following inductive lemma

$$\forall x(P(x) \to P(s(x))) \vdash \forall n \forall x(P(x) \to P(f(x,n))).$$

A proof φ of this lemma is

$$\dfrac{\begin{array}{cc}(\varphi_1) & (\varphi') \\ \vdash \forall x P(x) \to P(f(x,0))) & \forall x(P(x) \to P(s(x))), \forall x(P(x) \to P(f(x,0))) \vdash \forall n \forall x(P(x) \to P(f(x,n)))\end{array}}{\forall x(P(x) \to P(s(x))) \vdash \forall n \forall x(P(x) \to P(f(x,n)))}\ \mathit{cut}$$

*Partially supported by FWF project I-5848-N and recipient of an APART-MINT Fellowship of the Austrian Academy of Sciences at the Institute of Logic and Computation of the TU Wien.

†Partially supported by FWF project I-5848-N.

Cynthia Kop and Helida Salles Santos (Eds); 19th Workshop on
Logical and Semantic Frameworks with Applications (LSFA'24)
EPTCS 421, 2025, pp. 98–111, doi:10.4204/EPTCS.421.6

where φ' is

$$\dfrac{\dfrac{\dfrac{(\varphi_2)}{\forall x(P(x)\to P(f(x,0)))\vdash \forall x(P(x)\to P(f(x,0)))} \qquad \dfrac{(\varphi_3)}{\forall x(P(x)\to P(s(x))),\forall x(P(x)\to P(f(x,k)))\vdash \forall x(P(x)\to P(f(x,s(k))))}\,ind}{\forall x(P(x)\to P(f(x,0))),\forall x(P(x)\to P(s(x)))\vdash \forall x(P(x)\to P(f(x,l)))}\,\forall_r}{\forall x(P(x)\to P(f(x,0))),\forall x(P(x)\to P(s(x)))\vdash \forall n\forall x(P(x)\to P(f(x,n)))}\,\forall_r$$

and φ_1, φ_2, and φ_3 are proofs without induction rules. Now we define π as

$$\dfrac{\dfrac{(\varphi)}{\forall x(P(x)\to P(s(x)))\vdash \forall n\forall x(P(x)\to P(f(x,n)))} \qquad \dfrac{(\pi_1)}{S_\pi}}{\forall x(P(x)\to P(s(x)))\vdash \forall n\forall x((P(f(x,n))\to P(g(x,n)))\to (P(x)\to P(g(x,n))))}\,cut$$

π_1 is a proof without induction of the form

$$\dfrac{\dfrac{\dfrac{(\pi_1')}{P(u)\to P(f(u,i))\vdash (P(f(u,i))\to P(g(u,i)))\to (P(u)\to P(g(u,i)))}\,\forall_l^*}{\forall n\forall x(P(x)\to P(f(x,n)))\vdash (P(f(u,i))\to P(g(u,i)))\to (P(u)\to P(g(u,i)))}\,\forall_l^*}{\forall n\forall x(P(x)\to P(f(x,n)))\vdash \forall n\forall x((P(f(x,n))\to P(g(x,n)))\to (P(x)\to P(g(x,n))))}\,\forall_r^*$$

Using Gentzen's method of cut-elimination, we locate the place in the proof where $\forall n$ is introduced. In π_1 $\forall n\forall x(P(x)\to P(f(x,n)))$ is obtained from $\forall x(P(x)\to P(f(x,i)))$ by $\forall_l$. In φ we may delete the $\forall_r$ inference yielding the cut-formula and replace l by i. But in the attempt to eliminate the formula $\forall x(P(x)\to P(f(x,i)))$ in φ we get stuck, as we cannot cross the ind rule. Note that also ind cannot be eliminated as i is a variable. This problem is not due to the specific form of φ nor of ind. In fact, there exists no proof of S in **LK** *with only atomic cuts, even if ind is used. Induction on the formula*

$$\forall n\forall x((P(f(x,n))\to P(g(x,n)))\to (P(x)\to P(g(x,n))))$$

fails. To prove the end-sequent an inductive lemma is needed, i.e. something which implies $\forall n\forall x(P(x)\to P(f(x,n)))$ and cannot be eliminated.

There are methods for performing cut-elimination in presence of induction [2, 11], however, the resulting proofs do not have the subformula property and Herbrand's theorem cannot be realized. If induction is represented via schemata of proofs (see e.g. [4, 9]) schematic cut-elimination methods can be defined which allow the extraction of so-called *Herbrand schemata*, i.e. a generalization of Herbrand's theorem to schematic proofs. The underlying cut-elimination method is *schematic Ceres* [4, 9, 10], and in [1] the method was successfully applied to Fürstenberg's proof of the infinitude of primes, which is a complicated proof using topological concepts (see [5]). Fürstenberg's proof was formalized as a first-order schema, i.e. a sequence of proofs indexed by the number of primes assumed to exist, and schematic Ceres was applied to the entire sequence. The analysis was performed in a semi-automated way, and major parts of the analysis had to be performed by hand. Nevertheless, the analysis showed that from Fürstenberg's proof Euclid's elementary proof could be obtained. Though a fully automated analysis of this proof is not yet within reach, this example reveals the need for the development of a formal language for analyzing proofs with induction.

An open question has been the relation of proof schemata to systems of arithmetic. In particular it was not known whether the classes of proofs specifiable in primitive recursive arithmetic and via proof schemata coincide. In [8] it was shown that quantifier-free proofs in primitive recursive arithmetic (quantifier-free PRA-proofs) can be simulated by the proof schema formalism. In this paper we generalize and extend the results from [8] by considering also cases where quantifiers occur in PRA-proofs. We demonstrate that the proof in Example 1 cannot only be formalized as a proof schema, but also its schematic Herbrand sequent can be computed.

$$\frac{}{A \vdash A}\ Axiom \qquad \frac{\Gamma \vdash \Delta, F \qquad \Sigma, F \vdash \Pi}{\Gamma, \Sigma \vdash \Delta, \Pi}\ cut$$

$$\frac{\Gamma \vdash \Delta}{F, \Gamma \vdash \Delta}\ w_l \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, F}\ w_r$$

$$\frac{F, F, \Gamma \vdash \Delta}{F, \Gamma \vdash \Delta}\ c_l \qquad \frac{\Gamma \vdash \Delta, F, F}{\Gamma \vdash \Delta, F}\ c_r$$

$$\frac{\Gamma, F, G \vdash \Delta}{\Gamma, F \wedge G \vdash \Delta}\ \wedge_l \qquad \frac{\Gamma \vdash \Delta, F \qquad \Sigma \vdash \Pi, G}{\Gamma, \Sigma \vdash \Delta, \Pi, F \wedge G}\ \wedge_r$$

$$\frac{\Gamma, F \vdash \Delta \qquad \Sigma, G \vdash \Pi}{\Gamma, \Sigma, F \vee G \vdash \Delta, \Pi}\ \vee_l \qquad \frac{\Gamma \vdash \Delta, F, G}{\Gamma \vdash \Delta, F \vee G}\ \vee_r$$

$$\frac{\Gamma \vdash \Delta, F}{\Gamma, \neg F \vdash \Delta}\ \neg_l \qquad \frac{\Gamma, F \vdash \Delta}{\Gamma \vdash \Delta, \neg F}\ \neg_r$$

$$\frac{\Gamma \vdash \Delta, F \qquad \Sigma, G \vdash \Pi}{\Gamma, \Sigma, F \to G \vdash \Delta, \Pi}\ \to_l \qquad \frac{\Gamma, F \vdash \Delta, G}{\Gamma \vdash \Delta, F \to G}\ \to_r$$

$$\frac{\Gamma, F[t/x] \vdash \Delta}{\Gamma, \forall x F[x] \vdash \Delta}\ \forall_l \qquad \frac{\Gamma \vdash F[y/x], \Delta}{\Gamma \vdash \forall F, \Delta}\ \forall_r$$

$$\frac{\Gamma, F[y/x] \vdash \Delta}{\Gamma, \exists F \vdash \Delta}\ \exists_l \qquad \frac{\Gamma \vdash F[t/x], \Delta}{\Gamma \vdash \exists F, \Delta}\ \exists_r$$

Figure 1: The rules for **LK**. In the rule *Axiom*, A is quantifier free. In the rules $\forall_r$ and $\exists_l$ the variable y, the eigenvariable, cannot occur free in the lower sequent. In the rules $\forall_l$ and $\exists_r$ t is an arbitrary term. Note that we use a restricted form of **LK** as we do not allow the application of $\forall_r$ and $\exists_l$ on induction variables.

2 Schematic Language and PRA

In [10] and [3] schematic first-order languages were defined as a basis for inductive proof analysis. These languages are based on primitive recursive definitions of function symbols, predicate symbols and proofs. In this paper we focus on the language of primitive recursive arithmetic **PRA** as defined in [6] where we have only the equality as predicate symbol but infinitely many function symbols. The theory contains quantifier-free axioms and an induction rule with quantifier-free induction formulas.

We define the respective calculus as Gentzen's **LK** (see Figure 1), extended by an equational theory, and incorporating the following induction rule:

$$\frac{\Gamma \vdash \Delta, F(0) \qquad \Gamma, F(y) \vdash \Delta, F(y+1)}{\Gamma \vdash \Delta, F(n)}\ ind$$

where y and n are variables of sort ω (the type of the natural numbers). y does not occur in $\Gamma, \Delta, F(0)$ and F is quantifier-free. Note that in [6], the induction variable is an arbitrary term. Our restriction to a variable of sort ω is equivalent, as it is easy to see that any arbitrary term can be simulated in the conclusion. The calculus resulting from the combination of the rules from Figure 1, an equational theory $\mathscr{E}$ and *ind* is denoted by **PRA**.

The language is schematic in the sense that, in addition to successor s and constant $\bar{0}$, it contains a countably infinite set of function symbols $\mathscr{F}$ (of arbitrary arity) together with primitive recursive definitions of functions based on these symbols. So if g, h are in $\mathscr{F}$, h is n – ary and g is $n+2$ – ary we can

choose an $n+1$ – ary symbol f and add a set of two equations

$$\mathscr{E}(f) = \{f(\vec{x},\bar{0}) = h(\vec{x}),\ f(\vec{x},s(y)) = g(\vec{x},y,f(\vec{x},y))\}.$$

The function symbols in $\mathscr{F}$ have to be partially ordered such that in any definition $\mathscr{E}(f)$ as defined above $h < f$ and $g < f$. With $\mathscr{E}$ we denote the union of the sets $\mathscr{E}(f)$ for $f \in \mathscr{F}$. The minimal elements in $\mathscr{F}$ must fulfill the condition that $h(\vec{x})$ and $g(\vec{x},y,z)$ can be identified with terms over the signature $\{s,\bar{0}\}$. As an example we may consider the definition of $+$ and $*$ via two binary function symbols f and g:

$$\begin{array}{rcl} \mathscr{E}(f) & = & \{f(x,\bar{0}) = x,\ f(x,s(y)) = s(f(x,y))\}, \\ \mathscr{E}(g) & = & \{g(x,\bar{0}) = \bar{0},\ g(x,s(y)) = f(x,g(x,y))\}. \end{array}$$

Here $f < g$ and f is minimal. The characteristic feature of **PRA** is the fact that every ground term in this language evaluates to a *numeral*; numerals are elements of the form $s^n(\bar{0})$ for $n \in \mathbb{N}$, the set of all numerals is denoted by *Num*. We introduce the concept of *parameter* to classify a subset of the first-order variables V which are supposed to be evaluated. Thereby we fix the domain of interpretation for **PRA** to the standard model of arithmetic with domain $\mathbb{N}$ and 0 for $\bar{0}$ and successor for s. Parameters will play the role of induction eigenvariables in proofs. The set of parameters is denoted by $\mathscr{P}$. As an example, a schematic version of Example 1 would consider k as a parameter and the other variables as "ordinary" first-order variables. For proof schemata to be defined in Section 3 the parameters will be crucial in defining the semantics of schemata.

Definition 1 (parameter assignment)**.** *A* parameter assignment σ *is a mapping* $\mathscr{P} \to Num$ *with the following extensions to terms and formulas:*

- $\sigma(x) = x$ *for* $x \in V \setminus \mathscr{P}$,
- $\sigma(\bar{0}) = \bar{0},\ \sigma(s(t)) = s(\sigma(t))$,
- *For an* n *– ary* $f \in \mathscr{F}$ *we have* $\sigma(f(t_1,\ldots,t_n)) = f(\sigma(t_1),\ldots,\sigma(t_n))$,
- $\sigma(\neg A) = \neg\sigma(A)$,
- $\sigma(A \circ B) = \sigma(A) \circ \sigma(B)$ *for* $\circ \in \{\wedge,\vee,\to\}$,
- $\sigma(Qx.A) = Qx.\sigma(A)$ *if* $x \in V \setminus \mathscr{P}$ *and* $Q \in \{\forall,\exists\}$.

The set of all parameter assignments is denoted by $\mathscr{S}$.

As already mentioned every term containing parameters can be evaluated under parameter assignments; terms containing only parameters and no variables in $V \setminus \mathscr{P}$ evaluate to numerals, others are merely *partially evaluated*.

Definition 2 (evaluation)**.** *Let* f *be an* $(n+1)$ *– ary function symbol in* $\mathscr{F}$ *with* $n \in \mathscr{P}$ *and*

$$\mathscr{E}(f) = \{f(\vec{x},\bar{0}) = h(\vec{x}),\ f(\vec{x},s(n)) = g(\vec{x},y,f(\vec{x},n))\} \text{ and}$$

$t_1,\ldots,t_l$ *be terms,* $\vec{t} = (t_1,\ldots,t_l)$ *and* $k \in Num$. *Then,*

- $f(\vec{t},\bar{0})\downarrow = h(\vec{t})\downarrow$,
- $f(\vec{t},s(\bar{k}))\downarrow = g(\vec{t},\bar{k},f(\vec{t},\bar{k})\downarrow)\downarrow$,
- $r\downarrow = r$ *if* r *is a term not containing a symbol in* $\mathscr{F}$.

The evaluation operator has to be applied recursively to all subterms of a term containing symbols in $\mathscr{F}$. Assume that σ is a parameter assignment and t is a term with $\sigma(t)\downarrow = \bar{k}$ and $r \in \mathscr{F}$ then $\sigma(r(\vec{s},t))\downarrow = r(\sigma(\vec{s}))\downarrow, \bar{k})\downarrow$.

The evaluation under σ can be extended to formulas homomorphically in an obvious way.

Example 2. *Let $x,n \in V$, $x \in V \setminus \mathscr{P}$ and $n \in \mathscr{P}$; assume that $\sigma(n) = \bar{1}$. Let $f \in \mathscr{F}$ and*

$$\mathscr{E}(f) = \{f(x,\bar{0}) = x,\ f(x,s(n)) = s(f(x,n))\}.$$

Then $\sigma(f(x,s(n)))\downarrow = f(\sigma(x)\downarrow, \sigma(s(n))\downarrow) = f(x,\bar{2})\downarrow$ by $\sigma(x) = x$ and

- *$\sigma(s(n))\downarrow = \sigma(s(n)) = s(\sigma(n)) = s(\bar{1}) = \bar{2}$.*

Furthermore

$$f(x,\bar{2})\downarrow = s(f(x,\bar{1}))\downarrow = s(s(f(x,\bar{0})))\downarrow = s(s(x))$$

and so $\sigma(f(x,s(n))\downarrow = s(s(x))$. We see that by distinguishing parameters and variables we obtain just a partial evaluation, i.e. terms need not evaluate to numerals but just to other terms containing variables.

Let $A = \forall x.f(x,n) = f(n,x)$ and $\sigma(n)$ as above. Then

$$\sigma(A)\downarrow = \forall x.s(x) = f(\bar{1},x).$$

We also see that $\{\sigma(A) \mid \sigma \in \mathscr{S}\}$ defines the infinite set of formulas $\{\forall x.s^n(x) = f(\bar{n},x) \mid n \in \mathbb{N}\}$. So we can consider the formula A as a formula schema *describing an infinite sequence of formulas*

3 Proof Schema

The general idea of a proof schema is to represent a proof containing induction by a finite description of an infinite sequence of proofs without induction inferences: Assume a proof φ of the end-sequent $\vdash \forall x A(x)$ that uses an induction inference. Instead of considering the proof φ, we instead consider the infinite sequence of proofs φ_0, φ_1, φ_2, ... of end-sequents $\vdash A(\bar{0})$, $\vdash A(\bar{1})$, $\vdash A(\bar{2})$ ... The task is to find a finite description of this infinite sequence of proofs, the proof schema. A proof schema always represents a parameterized sequence, and an evaluation under a parameter assignment $\bar{n}$ results in the proof φ_n of $\vdash A(\bar{n})$. The underlying problem, that initially lead to the development of proof schemata, is to be able to analyze inductive proofs and extract their Herbrand sequents. Indeed, each of the proofs φ_0, φ_1, φ_2, ... is a simple **LK**-proof without induction inferences, and thus enjoys cut-elimination resulting in an analytic proof. The concept of proof schema was initially introduced in [4, 9] to address schemata involving a single parameter. Later, it was expanded to accommodate an arbitrary number of induction parameters [10].

Formally, proof schemata are constructed using proofs in an extension of **LK** by an equational theory. First, let us define the concept of schematic sequents (the definitions below are from [10]).

Definition 3 (schematic sequents)**.** *A schematic sequent is a sequent of the form $F_1,\ldots,F_\alpha \vdash G_1,\ldots,G_\beta$ where the F_i and G_j for $1 \leq i \leq \alpha$ and $1 \leq j \leq \beta$ are formula schemata. Let $S\colon F_1,\ldots,F_\alpha \vdash G_1,\ldots,G_\beta$ be a schematic sequent and σ a parameter assignment. Then the evaluation of S under σ is $\sigma(S)\downarrow\colon \sigma(F_1)\downarrow$ $,\ldots,\sigma(F_\alpha)\downarrow \vdash \sigma(G_1)\downarrow,\ldots,\sigma(G_\beta)\downarrow$.*

Definition 4. *Let $\mathscr{E}$ be an equational theory. We extend the calculus* **LK** *by the $\mathscr{E}$ inference rule $\dfrac{S(t)}{S(t')}\mathscr{E}$ where the term or input term schema t in the schematic sequent S is replaced by a term or input term schema t' where $t = t'$ is an instance of an equation in $\mathscr{E}$.*

The definitions below will use the schematic standard axiom set $\mathscr{A}_s$.

Definition 5 (schematic standard axiom set)**.** *Let $\mathscr{A}_s$ be the smallest set of schematic sequents that is closed under substitution containing all sequents of the form $A \vdash A$ for arbitrary atomic formula schemata A. Then $\mathscr{A}_s$ is called the schematic standard axiom set.*

Schematic derivations can be understood as parameterized sequences of **LK**-derivations where new kinds of axioms in the form of labeled sequents are included. These labeled sequents serve the purpose to establish recursive call structures in the proof. For constructing schematic derivations we introduce a countably infinite set Δ of *proof symbols* which are used to label the individual proofs of a proof schema. A proof schema uses only a finite set of proof symbols $\Delta^* \subset \Delta$. We assign an arity $A(\delta)$ to every $\delta \in \Delta^*$, $A(\delta)$ is the arity of the input parameters for the proof labeled by δ. Also, we need a concept of proof labels which serve the purpose to relate some leafs of the proof tree to recursive calls.

Definition 6 (proof label)**.** *Let $\delta \in \Delta$ and ϑ be a parameter substitution. Then the pair (δ, ϑ) is called a proof label.*

Definition 7 (labeled sequents and derivations)**.** *Let S be a schematic sequent and (δ, ϑ) a proof label, then $(\delta, \vartheta)\colon S$ is a* labeled sequent*. A* labeled *derivation is a derivation π where all leaves are labeled.*

In the definition below we will define a proof schema over a base-case proof (for parameter 0) and a step-case proof (for parameter $m+1$), where initial sequents are either axioms, or end-sequents from previously defined base- or step-case proofs. In general, the step-case proof for some proof symbol δ uses as initial sequent its own end-sequent, but under a parameter assignment m. Evaluating a schematic derivation means that initial sequents, which are no axioms, have to be replaced by their derivations.

Definition 8 (parameter replacement)**.** *Let $\vec{m}, \vec{n}$ be tuples of parameters. A parameter replacement on $\vec{n}$ with respect to $\vec{m}$ is a replacement substituting every parameter p in $\vec{n}$ by a term t_p, where the parameters of $t_p \in T^\omega$ are in $\vec{m}$.*

Definition 9 (schematic deduction and proof schema)**.** *Let $\mathscr{D}$ be the tuple $(\delta_0, \Delta^*, \Pi)$. $\mathscr{D}$ is called a schematic deduction from a finite set of schematic sequents $\mathscr{S}$ if the following conditions are fulfilled:*

- *Δ^* is a finite subset of Δ.*
- *$\delta_0 \in \Delta^*$, and $\delta_0 > \delta'$ for all $\delta' \in \Delta^*$ such that $\delta' \neq \delta_0$. δ_0 is called the main symbol.*
- *To every $\delta \in \Delta^*$ we assign a parameter tuple $\vec{n}_\delta$ of pairwise different parameters (called the passive parameters), and a parameter m_δ (called the active parameter).*
- *Π is a set of pairs $\{(\Pi(\delta, \vec{n}_\delta, m_\delta), S(\delta, \vec{n}_\delta, m_\delta)\}$, where $S(\delta, \vec{n}_\delta, m_\delta)$ is a schematic sequent, and*

$$\Pi(\delta, \vec{n}_\delta, m_\delta) = \{(\delta, \vec{n}_\delta, m_\delta) \to \rho(\delta, \vec{n}_\delta, m_\delta)\},$$

where $\rho(\delta, \vec{n}_\delta, 0) = \rho_0(\delta, \vec{n}_\delta)$, and $\rho(\delta, \vec{n}_\delta, s(m_\delta)) = \rho_1(\delta, \vec{n}_\delta, m_\delta)$, and there exists a (possibly empty) finite set of schematic sequents $\mathscr{C}(\delta)$ such that

1. *$\rho_0(\delta, \vec{n}_\delta)$ is a deduction of $S(\delta, \vec{n}_\delta, 0)$ from $\mathscr{S} \cup \mathscr{C}(\delta)$,*
2. *$\rho_1(\delta, \vec{n}_\delta, m_\delta)$ is a deduction of $S(\delta, \vec{n}_\delta, m_\delta + 1)$ from $\{(\delta, \Psi)\colon S(\delta, \vec{n}_\delta, m_\delta)\} \cup \mathscr{S} \cup \mathscr{C}(\delta)$, where (δ, Ψ) is a label, and Ψ the empty parameter replacement,*
3. *for all $S' \in \mathscr{C}(\delta)$, $S' = (\delta', \Psi)\colon S(\delta', \vec{n}_{\delta'}, m_{\delta'})\Psi$ where (δ', Ψ) is a label, $\delta' \in \Delta^*$ with $\delta > \delta'$ and Ψ is a parameter replacement on $(\vec{n}_{\delta'}, m_{\delta'})$ w.r.t. $(\vec{n}_\delta, m_\delta)$ such that the conditions* 1. *and* 2. *hold for δ' .*

If $\mathscr{S} = \mathscr{A}_s$ we call $\mathscr{D}$ a proof schema *of $S(\delta_0, \vec{n}_{\delta_0}, m_{\delta_0})$.*

In the example below we will formalize the PRA-proof from Example 1 as a proof schema.

Example 3. *In this example we will construct a proof schema of the end-sequent*

$$\forall x(P(x) \to P(s(x))) \vdash \forall x(P(f(x,n)) \to P(g(x,n))) \to (P(x) \to P(g(x,n)))$$

where

$$f(x,0) = x, \quad f(x,s(n)) = s(f(x,n)).$$

Note that this end-sequent is equivalent over the standard model to this in Example 1 for which no Herbrand sequent could be obtained. The only difference is that the $\forall n$ in the end-sequent of Example 1 disappears in the proof schema formalism, as n will be the schema's inductive parameter and we do not allow strong quantification of inductive parameters. Note further that in this example we consider only one parameter, but in general, we allow arbitrarily many. To start, let us first define a proof schema

$$\mathscr{D}_1 = \{(\delta, \rho(\delta,0), \rho(\delta,n+1))\},$$

with end-sequent

$$S(\delta)\colon \forall x(P(x) \to P(s(x))) \vdash \forall x(P(x) \to P(f(x,n))).$$

We define $\rho(\delta,0)$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{P(f(a,0)) \vdash P(f(a,0))}{P(a) \vdash P(f(a,0))}\,\mathscr{E}}{\vdash P(a) \to P(f(a,0))}\,\to_r}{\vdash \forall x(P(x) \to P(f(x,0)))}\,\forall_r}{\forall x(P(x) \to P(s(x))) \vdash \forall x(P(x) \to P(f(x,0)))}\,w_l$$

$\rho(\delta,n+1)$ is defined as follows:

$$\dfrac{(\delta,\emptyset)\colon S(\delta) \qquad (1)}{\forall x(P(x) \to P(s(x))) \vdash \forall x(P(x) \to P(f(x,n+1)))}\,cut, c_l$$

where (1) is

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(a) \vdash P(a) \qquad \dfrac{\dfrac{P(f(a,n)) \vdash P(f(a,n)) \qquad \dfrac{P(f(a,n+1)) \vdash P(f(a,n+1))}{P(s(f(a,n))) \vdash P(f(a,n+1))}\,\mathscr{E}}{P(f(a,n)), P(f(a,n)) \to P(s(f(a,n))) \vdash P(f(a,n+1))}\,\to_l}{P(f(a,n)), \forall x(P(x) \to P(s(x))) \vdash P(f(a,n+1))}\,\forall_l}{P(a), P(a) \to P(f(a,n)), \forall x(P(x) \to P(s(x))) \vdash P(f(a,n+1))}\,\to_l}{P(a) \to P(f(a,n)), \forall x(P(x) \to P(s(x))) \vdash P(a) \to P(f(a,n+1))}\,\to_r}{\forall x(P(x) \to P(f(x,n))), \forall x(P(x) \to P(s(x))) \vdash P(a) \to P(f(a,n+1))}\,\forall_l}{\forall x(P(x) \to P(f(x,n))), \forall x(P(x) \to P(s(x))) \vdash \forall x(P(x) \to P(f(x,n+1)))}\,\forall_r$$

Now we construct the proof schema

$$\mathscr{D} = \{(\delta', \rho(\delta',n), \rho(\delta',n+1))\} \cup \mathscr{D}_1,$$

with the end-sequent

$$S(\delta')\colon \forall x(P(x) \to P(s(x))) \vdash \forall x(P(f(x,n)) \to P(g(x,n))) \to (P(x) \to P(g(x,n))),$$

and $\delta' > \delta$. There is no internal recursion in δ' needed, hence we only define $\rho(\delta',n)$ as follows:

$$\frac{(\delta,\emptyset)\colon S(\delta) \qquad (2)}{\forall x(P(x)\to P(s(x)))\vdash \forall x(P(f(x,n))\to P(g(x,n)))\to (P(x)\to P(g(x,n)))}\ cut$$

where (2) *is*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(f(c,n))\vdash P(f(c,n)) \qquad P(g(c,n))\vdash P(g(c,n))}{P(f(c,n))\to P(g(c,n)),P(f(c,n))\vdash P(g(c,n))}\to_l \quad P(c)\vdash P(c)}{P(c),P(f(c,n))\to P(g(c,n)),P(c)\to P(f(c,n))\vdash P(g(c,n))}\to_l}{P(f(c,n))\to P(g(c,n)),P(c)\to P(f(c,n))\vdash P(c)\to P(g(c,n))}\to_r}{P(c)\to P(f(c,n))\vdash (P(f(c,n))\to P(g(c,n)))\to (P(c)\to P(g(c,n)))}\to_r}{\forall x(P(x)\to P(f(x,n)))\vdash (P(f(c,n))\to P(g(c,n)))\to (P(c)\to P(g(c,n)))}\forall_l}{\forall x(P(x)\to P(f(x,n)))\vdash \forall x(P(f(x,n))\to P(g(x,n)))\to (P(x)\to P(g(x,n)))}\forall_r$$

Proof schemata define infinite sequences of proofs, and can be evaluated under parameter assignments.

Definition 10 (evaluation of proof schema)**.** *Let $\mathscr{D}=(\delta_0,\Delta^*,\Pi)$ be a proof schema, and σ a parameter assignment. In defining the evaluation of the proof schema, $\mathscr{D}\sigma\downarrow$, we proceed by double induction on the ordering of proof symbols and the assignments σ.*

- *Let δ_i be a minimal element in Δ^*.*
  1. $\sigma(m_{\delta_i})=0$.
     Then, by definition of a proof schema, $\rho_0(\delta_i,\vec{n}_{\delta_i})$ is a proof with **LK***-inferences and inferences for defined symbols that contain schematic sequents. Let $S_1,\dots S_n$ be all the schematic sequents in $\rho_0(\delta_i,\vec{n}_{\delta_i})$. Then the evaluation of $\rho_0(\delta_i,\vec{n}_{\delta_i})$ under σ is denoted by $\rho_0(\delta_i,\vec{n}_{\delta_i})\downarrow$ and obtained by replacing all $S_1,\dots,S_n$ in $\rho_0(\delta_i,\vec{n}_{\delta_i})$ by $\sigma(S_1)\downarrow,\dots,\sigma(S_n)\downarrow$ and omitting the inferences for defined symbols.*
  2. $\sigma(m_{\delta_i})=\alpha>0$.
     Evaluate all schematic sequents except the leaves $(\delta_i,\emptyset)\colon S(\delta_i,\vec{n}_{\delta_i},m_{\delta_i})$ under σ. Let $\sigma[m_{\delta_i}/\alpha-1]$ be defined as $\sigma[m_{\delta_i}/\alpha-1](p)=\sigma(p)$ for all $p\neq m_{\delta_i}$ and $\sigma[m_{\delta_i}/\alpha-1](m_{\delta_i})=\alpha-1$. Then we replace the labeled sequent $(\delta_i,\emptyset)\colon S(\delta_i,\vec{n}_{\delta_i},m_{\delta_i})$ by the proofs $\rho_0(\delta_i,\vec{n}_{\delta_i})\sigma[m_{\delta_i}/\alpha-1]\downarrow$ if $\alpha-1=0$ and by $\rho_1(\delta_i,\vec{n}_{\delta_i},m_{\delta_i})\sigma[m_{\delta_i}/\alpha-1]\downarrow$ if $\alpha-1>0$. The result is an **LK***-proof.*
- *$\delta_i\in\Delta^*$ is not minimal.*
  1. $\sigma(m_{\delta_i})=0$.
     Evaluate all schematic sequents except the labeled sequents of the form $(\delta',\Psi)\colon$ $S(\delta',\vec{n}_{\delta'},m_{\delta'})\Psi$ for $\delta_i>\delta'$ and the corresponding parameter replacement Ψ under σ. Then replace the labeled sequent $(\delta',\Psi)\colon S(\delta',\vec{n}_{\delta'},m_{\delta'})\Psi$ by the proof $\rho_0(\delta',\vec{n}_{\delta'})\Psi\sigma\downarrow$ if $\sigma(m_{\delta'})=0$ and by the proof $\rho_1(\delta',\vec{n}_{\delta'},m_{\delta'})\Psi\sigma\downarrow$ otherwise.
  2. $\sigma(m_{\delta_i})=\alpha>0$.
     As above, except for the labeled sequents $(\delta_i,\emptyset)\colon S(\delta_i,\vec{n}_{\delta_i},m_{\delta_i})$ which are replaced by the proof $\rho_0(\delta_i,\vec{n}_{\delta_i})\sigma[m_{\delta_i}/\alpha-1]$ if $\alpha-1=0$ and by the proof $\rho_1(\delta_i,\vec{n}_{\delta_i},m_{\delta_i})\sigma[m_{\delta_i}/\alpha-1]$ otherwise.

$\mathscr{D}\sigma\downarrow$ is defined as $\rho_0(\delta_0,\vec{n}_{\delta_0})\sigma\downarrow$ for the $<$-maximal symbol δ_0 if $\sigma(m_{\delta_0})=0$, and by $\rho_1(\delta_0,\vec{n}_{\delta_0},m_{\delta_0})\sigma\downarrow$ if $\sigma(m_{\delta_0})>0$.

We will illustrate the evaluation of a proof schema for a concrete parameter in the example below.

Example 4. *Let $\mathscr{D}$ be the proof schema as defined in Example 3, and $\sigma(n) = 0$. Then $\mathscr{D}\sigma\downarrow$ is obtained by considering the derivation $\rho(\delta',0)$, where the proof call to $(\delta,\emptyset)\colon S(\delta)\{n \leftarrow 0\}$ is replaced by the derivation $\rho(\delta,0)$, hence we obtain*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(f(a,0)) \vdash P(f(a,0))}{P(a) \vdash P(f(a,0))}\,\mathscr{E}}{\vdash P(a) \to P(f(a,0))}\,\to_r}{\vdash \forall x(P(x) \to P(f(x,0)))}\,\forall_r}{\forall x(P(x) \to P(s(x))) \vdash \forall x(P(x) \to P(f(x,0)))}\,w_l \qquad (1)}{\forall x(P(x) \to P(s(x))) \vdash \forall x(P(f(x,0)) \to P(g(x,0))) \to (P(x) \to P(g(x,0)))}\,cut$$

where (1) *is*

$$\dfrac{\dfrac{\dfrac{\dfrac{P(c) \vdash P(c) \qquad \dfrac{P(f(c,0)) \vdash P(f(c,0)) \qquad P(g(c,0)) \vdash P(g(c,0))}{P(f(c,0)) \to P(g(c,0)), P(f(c,0)) \vdash P(g(c,0))}\,\to_l}{P(c), P(f(c,0)) \to P(g(c,0)), P(c) \to P(f(c,0)) \vdash P(g(c,0))}\,\to_l}{P(f(c,0)) \to P(g(c,0)), P(c) \to P(f(c,0)) \vdash P(c) \to P(g(c,0))}\,\to_r}{P(c) \to P(f(c,0)) \vdash (P(f(c,0)) \to P(g(c,0))) \to (P(c) \to P(g(c,0)))}\,\to_r}{\forall x(P(x) \to P(f(x,0))) \vdash (P(f(c,0)) \to P(g(c,0))) \to (P(c) \to P(g(c,0)))}\,\forall_l}{\forall x(P(x) \to P(f(x,0))) \vdash \forall x(P(f(x,0)) \to P(g(x,0))) \to (P(x) \to P(g(x,0)))}\,\forall_r$$

Now let us consider $\sigma(n) = 1$. Then $\mathscr{D}\sigma\downarrow$ is obtained by considering $\rho(\Delta',1)$ and replacing the proof link to $(\delta,\emptyset)\colon S(\delta)$ by the derivation $\rho(\delta,1)$. Note that in $\rho(\delta,1)$ (which in our formalism is denoted by $\rho(\delta,0+1)$) we have to replace the self-referencing proof link $(\delta,\emptyset)\colon S(\delta)$ by the derivation $\rho(\delta,0)$. Therefore, we obtain

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(f(a,0)) \vdash P(f(a,0))}{P(a) \vdash P(f(a,0))}\,\mathscr{E}}{\vdash P(a) \to P(f(a,0))}\,\to_r}{\vdash \forall x(P(x) \to P(f(x,0)))}\,\forall_r}{\forall x(P(x) \to P(s(x))) \vdash \forall x(P(x) \to P(f(x,0)))}\,w_l \qquad (1)}{\forall x(P(x) \to P(s(x))) \vdash \forall x(P(x) \to P(f(x,1)))}\,cut, c_l \qquad (2)}{\forall x(P(x) \to P(s(x))) \vdash \forall x(P(f(x,1)) \to P(g(x,1))) \to (P(x) \to P(g(x,1)))}\,cut$$

where (1) *is*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(a) \vdash P(a) \qquad \dfrac{\dfrac{P(f(a,0)) \vdash P(f(a,0)) \qquad \dfrac{P(f(a,1)) \vdash P(f(a,1))}{P(s(f(a,0))) \vdash P(f(a,1))}\,\mathscr{E}}{P(f(a,0)), P(f(a,0)) \to P(s(f(a,0))) \vdash P(f(a,1))}\,\to_l}{P(f(a,0)), \forall x(P(x) \to P(s(x))) \vdash P(f(a,1))}\,\forall_l}{P(a), P(a) \to P(f(a,0)), \forall x(P(x) \to P(s(x))) \vdash P(f(a,1))}\,\to_l}{P(a) \to P(f(a,0)), \forall x(P(x) \to P(s(x))) \vdash P(a) \to P(f(a,1))}\,\to_r}{\forall x(P(x) \to P(f(x,0))), \forall x(P(x) \to P(s(x))) \vdash P(a) \to P(f(a,1))}\,\forall_l}{\forall x(P(x) \to P(f(x,0))), \forall x(P(x) \to P(s(x))) \vdash \forall x(P(x) \to P(f(x,1)))}\,\forall_r$$

and (2) *is*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(f(c,1)) \vdash P(f(c,1)) \qquad P(g(c,1)) \vdash P(g(c,1))}{P(f(c,1)) \to P(g(c,1)), P(f(c,1)) \vdash P(g(c,1))} \to_l \quad P(c) \vdash P(c)}{P(c), P(f(c,1)) \to P(g(c,1)), P(c) \to P(f(c,1)) \vdash P(g(c,1))} \to_l}{P(f(c,1)) \to P(g(c,1)), P(c) \to P(f(c,1)) \vdash P(c) \to P(g(c,1))} \to_r}{P(c) \to P(f(c,1)) \vdash (P(f(c,1)) \to P(g(c,1))) \to (P(c) \to P(g(c,1)))} \to_r}{\forall x(P(x) \to P(f(x,1))) \vdash (P(f(c,1)) \to P(g(c,1))) \to (P(c) \to P(g(c,1)))} \forall_l}{\forall x(P(x) \to P(f(x,1))) \vdash \forall x(P(f(x,1)) \to P(g(x,1))) \to (P(x) \to P(g(x,1)))} \forall_r$$

In [10] the Skolemized version of the proof schema in Example 3 was analyzed and a so-called Herbrand schema was obtained. A Herbrand schema can be understood as a parameterized sequence of Herbrand instances for the quantified formulas occurring in the end-sequent. The reason for Skolemization lies in the schematic cut-elimination method. In this work we will not focus on this cut-elimination method or the proof analysis of schematic proofs, but only explain the general idea.

We start with a proof schema of a Skolemized end-sequent. This sequent has to be Skolemized as the proof transformation steps in the method would not be sound if eigenvariables were present. The proof schema is then split in two parts: In the first part we start with the formulas in the initial sequents that are ancestors of formulas in the end-sequent, and apply only the rules that operate on these formulas. By construction the thus obtained proof schema is cut-free, we call it the projection schema. In the second part, only the formulas in initial sequents that are ancestors of cut-formulas and the rules operating on these cut-ancestors are considered. By construction, as we only have formulas that are ancestors of cuts and are therefore cut out, we end up with the empty-sequent. Therefore, the initial sequents we started with are unsatisfiable and can be represented as an unsatisfiable schematic formula, which is called the characteristic formula schema. The characteristic formula schema can be refuted with a schematic refutation calculus defined in [3]. In [7] this calculus was improved and it was shown that the Herband instances of the refutation schema can be extracted in the form of Herbrand schemata. It was shown in [10] that the Herbrand schema from the refutation of the characteristic formula can be combined with the Herbrand schema from the projection schema to obtain the Herbrand schema of the original proof schema.

For the Skolemized version of Example 3, we obtain (after some simplifications) for the formula $\forall x(P(x) \to P(s(x)))$ the schematic Herbrand instances

$$\{x \leftarrow f(c,n)\},$$

where c just denotes a Skolem constant introduced when Skolemizing the end-sequent. Therefore, for some numeral α the instances are

$$c, s(c), s(s(c)), \ldots, s^{\alpha-1}(c).$$

4 Simulation of Primitive Recursive Arithmetic Through Proof Schemata

In this section, we will analyze the expressivity of proof schemata by showing that proof schemata simulate a restricted version of primitive recursive arithmetic, as defined in [6]. In [9], it was demonstrated that proof schemata are equivalent to a specific fragment of arithmetic known as *k-simple induction*. This variant restricts the introduction of new eigenvariables through induction. Recently, we extended the simulation to capture proof schemata that allow an arbitrary number of parameters [8]. In the latter work, we used a definition of primitive recursive arithmetic, as defined in [12], which does not admit

quantifier introduction. In this paper, we allow quantifier introduction with the restriction that we do not allow strong quantification of induction variables. However, we will later demonstrate that proof schemata can indeed capture certain proofs involving strongly quantified induction variables.

We will now show the translation from proof schemata to **PRA** and back. To do this, we note that every sequent $S : \Gamma \vdash \Delta$ corresponds to an equivalent formula $\mathscr{F}(S) := \bigvee \neg\Gamma \cup \Delta$.

Lemma 1. *Let $\mathscr{D}$ be a proof schema with end-sequent S. Then there exists a **PRA** proof of S.*

Proof. Let $\mathscr{D} = \{(\delta_i, \rho(\delta_i, \vec{n}_i, 0), \rho(\delta_i, \vec{n}_i, m_i+1)) \mid i \in \{1, ..., \alpha\}\}$ with $S(\delta_i) = S_i$ and if $i < j$ then $\delta_i > \delta_j$. Hence, $S(\delta_1) = S$.

We construct inductively **PRA** proofs of $\mathscr{F}(S_\gamma)$, starting with $\gamma = \alpha$. Assume we constructed **PRA** proofs $\xi_{\gamma+1}, ..., \xi_\alpha$ of $\mathscr{F}(S_{\gamma+1}), ..., \mathscr{F}(S_\alpha)$ respectively. Our aim is to construct a **PRA** proof of $\mathscr{F}(S_\gamma)$.

In $\rho(\delta_\gamma, \vec{n}_\gamma, 0)$ replace any proof call of the form $(\delta_j, \Psi) : S(\delta_j)\Psi$ by $\xi_j\Psi$ to obtain proof ξ^B_γ. Applications of $\neg^*_r$ will then yield a proof of $\vdash \mathscr{F}(S_\gamma)\{m_\gamma \leftarrow 0\}$.

In $\rho(\delta_\gamma, \vec{n}_\gamma, m_{\gamma+1})$ consider any branches that lead into a self-referencing proof call of the form $(\delta_\gamma, \emptyset) : S(\delta_\gamma)$. Note that any introduction of a quantifier in these branches is cut and will not be part of the end-sequent of this derivation. Otherwise, the proof call conditions are violated. Further note that $\rho(\delta_\gamma, \vec{n}_\gamma, m_{\gamma+1})$ is an **LK** derivation and therefore admits cut-elimination. Using this, we eliminate the introduction of all strong quantifiers in these branches and replace any self-referencing proof call of the form $(\delta_\gamma, \emptyset) : S(\delta_\gamma)$ by axiom $\mathscr{F}(S(\delta_\gamma)) \vdash \mathscr{F}(S(\delta_\gamma))$.

For all other branches, replace any proof call of the form $(\delta_j, \Psi) : S(\delta_j)\Psi$ with $j \neq \gamma$ by $\xi_j\Psi$ to obtain proof ξ^S_γ. Applications of $\neg_r$ and c_r will then yield a proof of $\mathscr{F}(S_\gamma)\{m_\gamma \leftarrow y\} \vdash \mathscr{F}(S_\gamma)\{m_\gamma \leftarrow y+1\}$.

The desired proof ξ_γ is then constructed as follows:

$$\dfrac{\dfrac{\xi^B_\gamma}{\vdash \mathscr{F}(S_\gamma)\{m_\gamma \leftarrow 0\}}\,\neg^*_r \qquad \dfrac{\xi^S_\gamma}{\mathscr{F}(S_\gamma)\{m_\gamma \leftarrow y\} \vdash \mathscr{F}(S_\gamma)\{m_\gamma \leftarrow y+1\}}\,\neg^*_r, c^*_r}{\vdash \mathscr{F}(S_\gamma)}\,\mathit{ind}$$

Note that in case of a proof call which includes an instantiation, we use *cut* instead of *ind*. Finally, we use cuts to derive S_γ from the proof of $\mathscr{F}(S_\gamma)$.

For proof tuples without internal recursion, it suffices to replace any non-self-referencing proof call with its respective **PRA** derivation; an application of *ind* is not necessary.

□

The introduction of quantifiers does not affect our translation of **PRA** proofs into proof schemata, as demonstrated in a previous paper. Therefore, we will simply present the proof from [8] here:

Lemma 2. *Let π be a **PRA** proof of S. Then there exists a proof schema with end-sequent S.*

Proof. Let π contain α induction inferences

$$\dfrac{\Gamma_\beta \vdash \Delta_\beta, F_\beta(0) \qquad \Gamma_\beta, F_\beta(y) \vdash \Delta_\beta, F_\beta(y+1)}{\Gamma_\beta \vdash \Delta_\beta, F_\beta(n_\beta)}\,\mathit{ind}$$

where $a \leq \beta \leq \alpha$. W.l.o.g. assume that if $\gamma < \beta$ then the induction inference with conclusion $\Gamma_\beta \vdash \Delta_\beta, F_\beta(n_\beta)$ is above the induction inference with conclusion $\Gamma_\gamma \vdash \Delta_\gamma, F_\gamma(n_\gamma)$. We define $\vec{n} = \{n_i \mid i \in \{1...\alpha\}\} \cup V(\pi)$ as the set of all induction variables, where n_i denotes the induction variable of the i-th

induction inference, together with the set of free variables and constants V in π. Let T be the transformation taking an **PRA** proof to a proof schema by replacing the induction inferences with conclusion $\Gamma_\gamma \vdash \Delta_\gamma, F_\gamma(n_\gamma)$ by a proof call $(\delta_\gamma, \{m \leftarrow n_\gamma\}) : S(\delta_\gamma)\{m \leftarrow n_\gamma\}$ with $S(\delta_\gamma) = \Gamma_\gamma \vdash \Delta_\gamma, F_\gamma(m)$.

We will inductively construct a proof schema $\mathscr{D} = \{(\delta_i, \rho(\delta_i, \vec{n}, 0), \rho(\delta_i, \vec{n}, m+1)) \mid i \in \{1...\alpha\}\}$ with end-sequent $S(\delta_i) = \Gamma_i \vdash \Delta_i, F_i(m+1)$ for each tuple and $\delta_i > \delta_{i+1}$. Assume we already constructed proof schema $\mathscr{D}_{\beta+1} = \{(\delta_i, \rho(\delta_i, \vec{n}, 0), \rho(\delta_i, \vec{n}, m+1)) \mid i \in \{(\beta+1)...\alpha\}\}$.

Consider the induction inference with conclusion $\Gamma_\beta \vdash \Delta_\beta, F_\beta(n_\beta)$. Let φ_1 be the derivation above the left premise and φ_2 be the derivation above the right premise. We construct a proof schema $\mathscr{D}_\beta = \{(\delta_\beta, \rho(\delta_\beta, \vec{n}, 0), \rho(\delta_\beta, \vec{n}, m+1))\} \cup \mathscr{D}_{\beta+1}$ with $\rho(\delta_\beta, \vec{n}, 0) = T(\varphi_1)$ and $\rho(\delta_\beta, \vec{n}, m+1) =$

$$\dfrac{\dfrac{(\delta_\beta, \emptyset) : S(\delta_\beta)}{\Gamma_\beta \vdash \Delta_\beta, F_\beta(m)} \quad \dfrac{T(\varphi_2)}{\Gamma_\beta, F_\beta(m) \vdash \Delta_\beta, F_\beta(m+1)}}{\Gamma_\beta \vdash \Delta_\beta, F_\beta(m+1)}\, cut, c_l^*, c_r^*$$

Summarising, $(\delta_\beta, \rho(\delta_\beta, \vec{n}, 0), \rho(\delta_\beta, \vec{n}, m+1))$ is a proof schema tuple with end-sequent $\Gamma_\beta \vdash \Delta_\beta, F_\beta(n)$, as desired.

Finally, the part of π located beneath the last induction inference is translated into proof schema $\mathscr{D}' = \{(\delta', \rho(\delta', \vec{n}, m), \rho(\delta', \vec{n}, m+1))\} \cup \mathscr{D}$ with $S(\delta') = S$ and $\delta' > \delta_i$ for $i \in \{1...\alpha\}$. Let φ be the derivation above S. As there is no internal recursion in δ' needed, we only define $\rho(\delta', \vec{n}, m) = \dfrac{T(\varphi)}{S}$. □

As proof schemata are intended to be evaluated for a specific parameter assignment, the above translation does not generally apply to the strong quantification of induction variables. Consider the following **PRA** derivation, and for simplicity, let φ_1 and φ_2 be induction-free:

$$\dfrac{\dfrac{(\varphi_1)}{\Gamma \vdash \Delta, F(0)} \quad \dfrac{(\varphi_2)}{\Gamma, F(y) \vdash \Delta, F(y+1)}}{\dfrac{\Gamma \vdash \Delta, F(n)}{\Gamma \vdash \Delta, \forall z F(z)}\, \forall : r}\, ind$$

A translation, as described in Lemma 2, would initially yield a proof schema $\mathscr{D} = (\delta, \rho(\delta, \vec{n}, 0), \rho(\delta, \vec{n}, m+1))$ with end-sequent $S(\delta) = \Gamma \vdash \Delta, F(m)$ for the induction. In the subsequent step, a strong quantification of parameter m is not feasible, as evaluating $\mathscr{D}$ would violate the eigenvariable condition in $\forall_r$.

However, there are instances where proof schemata can simulate strongly quantified induction variables in **PRA** proofs. If a quantified induction variable is subsequently cut in a proof, we can omit the application of the quantifier rule and shift the cut upwards in a Gentzen-style manner. To achieve this, we locate all applications of a weak quantifier rule that lead to the cut formula in the right branch of the proof and instantiate the parameter of the proof schemata representing the induction with the respective terms found in this way. This process is illustrated in the following example.

Example 5. *In this example we translate a **PRA** derivation with a strongly quantified induction variable into a proof schema.*

*Consider the following **PRA** derivation π with φ_1, φ_2, ψ_1 and ψ_2 induction free for simplicity. Note that the induction variable n of the left induction rule is strongly quantified and subsequently cut. Let π be:*

$$\dfrac{\dfrac{\dfrac{(\varphi_1)}{\Gamma \vdash \Delta, F(0)} \quad \dfrac{(\varphi_2)}{\Gamma, F(y) \vdash \Delta, F(y+1)}\, ind}{\Gamma \vdash \Delta, F(n)}\, \forall_r}{\Gamma \vdash \Delta, \forall z F(z)} \quad \dfrac{\dfrac{\dfrac{(\psi_1)}{\Sigma, F(t_1) \vdash \Pi, G(0)}\, \forall_l \quad \dfrac{(\psi_2)}{\Sigma, F(t_2), G(x) \vdash \Pi, G(x+1)}\, \forall_l}{\Sigma, \forall z F(z), G(x) \vdash \Pi, G(x+1)}\, ind}{\Sigma, \forall z F(z) \vdash \Pi, G(m)}\, cut$$

$$\Gamma, \Sigma \vdash \Delta, \Pi, G(m)$$

In the right branch of π, there are two applications of $\forall_l$. Note the respective terms t_1 and t_2, as we will later use them to instantiate the proof schema representing the induction in the left branch.

We construct a proof schema $\mathscr{D} = \{(\delta_1, \rho(\delta_i, \vec{n}, 0), \rho(\delta_i, \vec{n}, m_i + 1)) \mid i \in \{0,1,2\}\}$. For the left induction in π, we define $\rho(\delta_2, \vec{n}, 0) = \varphi_1$ and $\rho(\delta_2, \vec{n}, m_2 + 1)$ as

$$\dfrac{\dfrac{(\delta_2, \emptyset) : S(\delta_2)}{\Gamma \vdash \Delta, F(m_2)} \quad \dfrac{(\varphi_2)}{\Gamma, F(m_2) \vdash \Delta, F(m_2+1)}}{\Gamma \vdash \Delta, F(m_2+1)}\, cut, c_l, c_r$$

For the right induction in π, we omit the applications of $\forall_l$ and use w_l instead. We define $\rho(\delta_1, \vec{n}, 0)$ as

$$\dfrac{\dfrac{(\psi_1)}{\Sigma, F(t_1) \vdash \Pi, G(0)}}{\Sigma, F(t_1), F(t_2) \vdash \Pi, G(0)}\, w_l$$

and $\rho(\delta_1, \vec{n}, m_1 + 1)$ as

$$\dfrac{\dfrac{(\delta_1, \emptyset) : S(\delta_1)}{\Sigma, F(t_1), F(t_2) \vdash \Pi, G(m_1)} \quad \dfrac{\dfrac{(\psi_2)}{\Sigma, F(t_2), G(m_1) \vdash \Pi, G(m_1+1)}}{\Sigma, F(t_1), F(t_2), G(m_1) \vdash \Pi, G(m_1+1)}\, w_l}{\Sigma, F(t_1), F(t_2) \vdash \Pi, G(m_1+1)}\, cut,\ c_l,\ c_r$$

Lastly we define δ_0. As previously mentioned, we instantiate the proof schema δ_2 with the terms t_1 and t_2. This allows us to perform a cut on an instantiated formula rather than a quantified one, and we can omit the application of $\forall_r$. Since there is no internal recursion in δ_0, we only define $\rho(\delta_0, \vec{n}, m_0)$ as

$$\dfrac{\dfrac{\dfrac{(\delta_2, \{m_2 \leftarrow t_1\}) : S(\delta_2)\{m_2 \leftarrow t_1\}}{\Gamma \vdash \Delta, F(t_1)} \quad \dfrac{(\delta_2, \{m_2 \leftarrow t_2\}) : S(\delta_2)\{m_2 \leftarrow t_2\}}{\Gamma \vdash \Delta, F(t_2)}\, \wedge_r}{\Gamma \vdash \Delta, F(t_1) \wedge F(t_2)} \quad \dfrac{\dfrac{(\delta_1, \emptyset) : S(\delta_1)}{\Sigma, F(t_1), F(t_2) \vdash \Pi, G(m_1)}}{\Sigma, F(t_1) \wedge F(t_2) \vdash \Pi, G(m_1)}\, \wedge : l}{\Gamma, \Sigma \vdash \Delta, \Pi, G(m_1)}\, cut$$

5 Conclusion

It was shown in [10] that when a proof with induction is formulated as proof schema, a recursive structure that represents the proof's Herbrand sequent can be extracted. It however remained an open question to relate proof schemata to systems of arithmetic, i.e. to identify the class of proofs that can be represented as proof schemata and analyzed with the methods in [10]. A first investigation in this direction was presented in [8], where it was shown that proofs in quantifier-free primitive recursive arithmetic (quantifier-free PRA) can be represented as proof schemata, and that quantifier-free proof schemata can be translated back into quantifier-free PRA. In this work we generalize and extend this result by investigating also the cases for quantifiers in proofs. We demonstrate the translation in both directions, with the condition that the inductive parameter is not quantified.

Together with a completeness result for the proof analysis method in [10] (a result not obtained so far), the result in this paper will yield a realization of Herbrand's theorem for an expressive fragment of formal number theory.

References

[1] Matthias Baaz, Stefan Hetzl, Alexander Leitsch, Clemens Richter & Hendrik Spohr (2008): *CERES: An analysis of Fürstenberg's proof of the infinity of primes*. Theoretical Computer Science 403(2-3), pp. 160–175, doi:10.1016/j.tcs.2008.02.043.

[2] James Brotherston & Alex Simpson (2011): *Sequent calculi for induction and infinite descent*. Journal of Logic and Computation 21(6), pp. 1177–1216, doi:10.1093/logcom/exq052.

[3] David M. Cerna, Alexander Leitsch & Anela Lolic (2021): *Schematic Refutations of Formula Schemata*. Journal of Automated Reasoning 65(5), pp. 599–645, doi:10.1007/s10817-020-09583-8.

[4] Cvetan Dunchev, Alexander Leitsch, Mikheil Rukhaia & Daniel Weller (2013): *Cut-elimination and proof schemata*. In: International Tbilisi Symposium on Logic, Language, and Computation, Springer, pp. 117–136, doi:10.1007/978-3-662-46906-4_8.

[5] Hillel Fürstenberg (1955): *On the Infinitude of the Primes*. American Mathematical Monthly 62(5), p. 353, doi:10.2307/2307043.

[6] Jean Yves Girard (Napoli,1987): *Proof Theory and Logical Complexity*. 1, Biblopolis.

[7] Alexander Leitsch & Anela Lolic (2024): *Herbrand's Theorem in Inductive Proofs*. In: Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR, EPiC Series in Computing 100, EasyChair, pp. 295–310, doi:10.29007/dwdf.

[8] Alexander Leitsch, Anela Lolic & Stella Mahler (2024): *On Proof Schemata and Primitive Recursive Arithmetic*. In: LPAR 2024 Complementary Volume, Kalpa Publications in Computing 18, EasyChair, pp. 117–130, doi:10.29007/4g2q.

[9] Alexander Leitsch, Nicolas Peltier & Daniel Weller (2017): *CERES for first-order schemata*. Journal of Logic and Computation 27(7), pp. 1897–1954, doi:10.1093/logcom/exx003.

[10] Anela Lolic (2020): *Automated Proof Analysis by CERES*. Ph.D. thesis, Technical University of Vienna, doi:10.34726/hss.2020.47184.

[11] Raymond McDowell & Dale Miller (2000): *Cut-elimination for a logic with definitions and induction*. Theoretical Computer Science 232(1-2), pp. 91–119, doi:10.1016/S0304-3975(99)00171-1.

[12] Gaisi Takeuti (1987): *Proof Theory*. North Holland, second edition.

An Execution Model for RICE

Steven Libby
University of Portland, Portland OR 97203, USA
libbys@up.edu

In this paper, we build on the previous work of the RICE [23] compiler by giving its execution model. We show the restrictions to the FlatCurry language that were made to produce executable code, and present the execution model using operational semantics similar to Launchbury [21]. Finally, we show that the execution model conforms with the standard operational semantics for Curry [2].

1 Introduction

Recently there has been a renewed interest in the efficient execution of functional logic programs [7, 9, 10, 23]. This has proven to be a rich area of new ideas. We look in particular at the RICE Curry compiler, which has shown to produce efficient code [23]. This paper provides an execution model for RICE.

Previous work on this compiler showed the execution of programs using a translation to C code. While this gets the point across, it is difficult to reason about the correctness of the implementation. Instead we take an approach similar to Braßel [11]. We begin by showing the execution model for the RICE compiler, and show how it is consistent with the standard operational semantics for Curry [2]. The primary contribution of this paper is the execution model.

Our execution model differs form previous works in a few respects. First we differ from Albert et al. [2] by encoding the search strategy into the semantics itself. While Albert et al [1] parameterize their semantics on the search strategy, the implementation is left abstract. Braßel [11] gives a fully realized execution model for the Kics2 compiler, which implements non-determinism with pull tabbing [11]. Our work is specifically about the RICE compiler, which implements non-determinism using backtracking [23]. This is done primarily because Kics2's performance degrades on functions that are not right-linear [23]. Our aim is to give a clear understanding of the execution of programs compiled with RICE.

The rest of the paper is organized as follows. First, we discuss previous work in the execution of Curry programs and why we chose a different approach. Second, we examine the syntax of an intermediate representation of Curry, and restrict it to a form that provides more efficient execution. Third, we provide the semantics for our restricted code. Fourth, we show a correspondence with the original semantics. Finally, we discuss future work and conclude.

2 Background

Curry is a functional logic language; it has a syntax similar to Haskell, but extends the language in several ways. The two important differences for this work are the addition of non-determinism and free variables.

We can construct a non-deterministic expression using the choice operator (`?`). For example, in the following code, `pickOne` non-deterministically picks an element in a list, while `member` constrains the choice to determine membership in that list. If the element `x` is not in `l`, then the member function simply fails and does not returen a value, so there is no need for an `otherwise` case like in Haskell.

Cynthia Kop and Helida Salles Santos (Eds); 19th Workshop on
Logical and Semantic Frameworks with Applications (LSFA'24)
EPTCS 421, 2025, pp. 112–129, doi:10.4204/EPTCS.421.7

```
pickOne (x:xs) = x ? pickOne xs
member x l
  | x == pickOne l = True
```

We can also create free (or logic) variables using the syntax `while x free`. For example, we can create a similar `member` function by constraining a list with free variables to match our input list.

```
member x l
 | l == first++[x]++last = True
  where first, last free
```

Non-determinism can be implemented by picking a search strategy [12, 24], and free variables can be constrained through narrowing [5, 16].

Curry originally grew out of the field of term rewriting and narrowing. Antoy [3] showed that term rewrite systems belonging to the class of Inductively Sequential systems could be given an optimal rewriting strategy, which was later extended by Antoy et al. [5] to an optimal narrowing strategy. Later Antoy [4] showed how this class of rewriting systems could be extended to include overlapping rules, and these Limited Overlapping Inductively Sequential (LOIS) systems could be computed with a combination of narrowing and a search strategy. This narrowing plus search proved to be a solid basis for the Curry language and implementations [14].

While narrowing proves to be a strong theoretical foundation for Curry, there are implementation issues that need to be addressed. In particular, Curry is a lazy language that contains higher order functions. While there are theories of higher order rewriting [26], Curry implementations often tend to use defunctionalization to deal with higher order rewriting [12, 16], thus turning it back into a first order rewriting system. While this addresses the theoretical issues of higher order functions nicely, it is not efficient [9, 25]. It should be noted that the implementations of KiCS2 and MCC do not rely on defunctionalization [12, 24].

Lazy evaluation can also pose an issue with using term rewriting as a basis for Curry. Terms have a tree-like structure, and do not support sharing. If a variable is duplicated in a rewrite rule, the entire subterm is duplicated. On the other hand, lazy programs do not ever duplicate expressions. In lazy functional languages, this is important for efficiency reasons. However in functional logic languages, this changes the semantics. Therefore sharing must be preserved at all costs [17]. For this reason, Curry is often presented as a graph rewriting system [13].

Hanus et al. [2] described an intermediate language called FlatCurry in order to give a semantics for Curry. They then described a natural semantics for FlatCurry in the style of Launchbury [21]. However they did not specify how choice should be handled, instead leaving it up to the implementation. This initial Curry semantics was extended to be completely deterministic [1] by parameterising with a search strategy. Braßel [11] uses this semantics as a starting point. He then introduces several transformations to the FlatCurry program to put it in a form that can be readily translated to Haskell. Our work is similar to Braßel's work. We recall the syntax for FlatCurry, describe the changes we made, and give the semantics for our new version of FlatCurry. Our work differs from previous approaches because we encode the search strategy into the evaluation itself. This makes for a more complicated semantics, but it allows us to examine the efficiency of Curry programs as a whole. This can lead to new optimizations like fast backtracking [22, 23].

f	$\Rightarrow$	$\mathrm{f}\ \overline{\mathrm{x}} = e$	function definition
e	$\Rightarrow$	x	Variable
	$\mid$	$e_1\ ?\ e_2$	Choice
	$\mid$	$\bot$	Failed
	$\mid$	$\mathrm{f}\ \overline{e}$	Function Application
	$\mid$	$\mathrm{C}\ \overline{e}$	Constructor Application
	$\mid$	**let** $\overline{x = e}$ **in** e	Variable Declaration
	$\mid$	**let** $\overline{x}$ **free in** e	Free Variable
	$\mid$	**case** e **of** $\overline{p \rightarrow e}$	Case Expression
p	$\Rightarrow$	$\mathrm{C}\ \overline{x}$	Constructor Pattern
	$\mid$	l	Literal Pattern

Figure 1: The syntax of FlatCurry We use the convention of x for variables, `f` for function names, and `C` for constructor names.

3 FlatCurry

We begin with a discussion of the abstract syntax of FlatCurry [2]. The language itself is similar to Core Haskell with a few important alterations to support the features of functional logic programming. The syntax is given in Figure 1. The semantics for FlatCurry are recalled in Figure 2. In this semantics, free variables are represented as a variable that is mapped to itself in the heap $\Gamma[x \mapsto x]$.

The most apparent difference is the addition of the choice operator ?, the free variable declaration **let** $\overline{x}$ **free in** e, and the $\bot$ constant. The choice operator and free declaration correspond to their counterparts in the Curry language. The $\bot$ constant represents a failed computation. As we will see, $\bot$ is propagated up through the computation until it is disregarded, or it reaches the root, at which point we say the computation fails.

Another peculiarity of this syntax is that there is no general form for application. In fact, that are not any lambda expressions in the language. However, this last part is only an apparent difference as the compiler has already completed lambda lifting [18] by the time it produces FlatCurry code.

The absence of a general application form is slightly more complicated. We can apply a function to arguments, but in this syntax there is no mechanism to apply a function to another function. This restriction would make it impossible to support higher order functions.

We solve this problem with a general `apply` function. In the expression $(\text{apply}\ e_1\ e_2)$ will evaluate e_1 to a function or partial application and apply it to e_2. In the theory, this is handled with defunctionalization [16], so it is left out of the natural semantics of Curry [2]. However, we will handle apply with the standard eval-apply method [25].

We also make a few changes from previous presentations of the syntax [2, 16]. We include failure as the value $\bot$. This allows us to encode failing computations explicitly in our execution model, and to show how failure propagates throughout a computation. We also have a separate construct for declaring free variables because the standard implementation of FlatCurry also has a separate construct.

Finally, literals, primitive operations, and residuation are all outside the scope of this paper, although not outside the scope of the compiler. This is done both for brevity and because their implementations are not novel.

(Nat-VarCons) $\Gamma[x \mapsto t] : x \Downarrow_C \Gamma[x \mapsto t] : t$ where t is a head normal form

(Nat-VarExp) $$\frac{\Gamma[x \mapsto e] : e \Downarrow_C \Delta : v}{\Gamma[x \mapsto e] : x \Downarrow_C \Gamma[x \mapsto v] : v}$$ where e is not a head normal form

(Nat-Val) $\Gamma : v \Downarrow_C \Gamma : v$ Where v is a head normal form

(Nat-Fun) $$\frac{\Gamma : \rho(e) \Downarrow_C \Delta : v}{\Gamma : f\ \overline{y} \Downarrow_C \Delta : v}$$ where $f\ \overline{x} = e \in P$ is a defined function and $\rho(x_n) = y_n$

(Nat-Let) $$\frac{\Gamma[\overline{y_k \mapsto \rho(e_k)}] : e \Downarrow_C \Delta : v}{\Gamma : \textbf{let}\ \overline{x_k = e_k}\ \textbf{in}\ e \Downarrow_C \Delta : v}$$ where $\rho(x_n) = y_n$

(Nat-Or) $$\frac{\Gamma : e_i \Downarrow_C \Delta : v}{\Gamma : e_1\ ?\ e_2 \Downarrow_C \Delta : v}$$ where $i \in \{1,2\}$

(Nat-Select) $$\frac{\Gamma : e \Downarrow_C \Delta : C_i(\overline{z}) \qquad \Delta : e_i[\overline{y \mapsto z}] \Downarrow_C \Theta : v}{\Gamma : \textbf{case}\ e\ \textbf{of}\ \overline{C_i(\overline{y}) \to e_i} \Downarrow_C \Theta : v}$$

(Nat-Guess) $$\frac{\Gamma : e \Downarrow_C \Delta : x \qquad \Delta[x \mapsto C_i(\overline{y})][\overline{y \mapsto y}] : e_i \Downarrow_C \Theta : v}{\Gamma : \textbf{case}\ e\ \textbf{of}\ \overline{C_i(\overline{y}) \to e_i} \Downarrow_C \Theta : v}$$

Figure 2: Natural semantics for Curry [2]
Following the conventions, $\Gamma[x \mapsto v]$ can be used to lookup or update variable x in heap Γ with value v.

Curry

```
and False _ = False
and True False = False
and True True = True
```

FlatCurry

```
and x y = case x of
              False -> False
              True -> case y of
                           False -> False
                           True -> True
```

restricted FlatCurry

```
and x y = case x of
              False -> False
              True -> and1 y

and1 y = case y of
             False -> False
             True -> True
```

Figure 3: A Curry function, a FlatCurry function, and a restricted FlatCurry function.

4 Restricted FlatCurry

We now turn our attention to transforming FlatCurry programs so that they are more amenable to an implementation. Our restricted language is very similar to Braßel's flat uniform programs [11], and is similar, although not identical to, A-Normal form [15]. The primary restrictions of Restricted FlatCurry are that functions and constructors are only applied to trivial arguments; let expressions cannot be nested; and each function definition can contain at most one case expression. We split the syntax into three sections: blocks, statements, and expressions. A block consists of a single `case` where each of the branches contains statements. A statement consists of zero or more `let` expressions, followed by an expression. Finally, an expression can be either a variable, literal, choice, failure, function application, constructor application, or `free`. We only allow a single case for each function, and all declarations must occur as early as possible. Furthermore, all applications, including function, constructor, and choice, must be applied to variables. The difference between the body, statements, and expressions is only for the purposes of giving structure to Curry functions. An example of a FlatCurry program, and a restricted FlatCurry program can be seen in Figure 3. Throughout the rest of the paper, we will refer to everything as an expression. This structure closely corresponds to the structure of ICurry [6].

We changed the representation of free variables in this syntax to correspond with their role in the RICE runtime. A free variable is a normal form that case expressions can narrow. It would be perfectly sensible to replace the free variable with a generator at this point [16], but RICE implements narrowing.

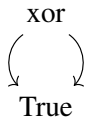
f	$\Rightarrow$	$f\,\overline{x} = b$	Function Definition
b	$\Rightarrow$	**case** x **of** $\overline{p \to s}$	Case Expression
	$\mid$	s	statement
s	$\Rightarrow$	**let** $\overline{x = e}$ **in** s	Variable Declaration
	$\mid$	e	Return Expression
e	$\Rightarrow$	x	Variable
	$\mid$	l	Literal
	$\mid$	$\bot$	Failed
	$\mid$	$x\,?\,x$	Choice
	$\mid$	**free**	Free variable
	$\mid$	$f\,\overline{x}$	Function Application
	$\mid$	$C\,\overline{x}$	Constructor Application
	$\mid$	apply $x\,\overline{x}$	Application
p	$\Rightarrow$	$C\,\overline{x}$	Constructor Pattern
	$\mid$	l	Literal Pattern

Figure 4: Syntax of restricted FlatCurry

5 Heap Representation

Now that we have a syntax for Curry, we can discuss the execution model. A Curry program consists of a set of functions as well as a single expression to evaluate. The expression is represented as a directed rooted graph that we will continually reduce. The graph plays the same role as the heap in traditional implementations of functional languages [8, 20]. We refer to it as a graph to stay closer in line with the theory of Curry.

The graph nodes are given in Figure 5. We use the notation $\mathrm{f}(x)$ to represent a function application to distinguish it from the FlatCurry syntax. Specifically, this represents the node f, with a single child x. If G is a graph with node n, then $G[n]$ refers to the subgraph rooted by node n, and $G[n \mapsto g]$ means replace node n in G with the graph g. We also use the convention that if a node n is referred to more than once, then it is shared. For example, if $G[n] = True$, then $xor(n,n)$ refers to the following graph.



We discuss the different nodes below. The graph contains seven different types of nodes: $\bot$, **free**, ?, function, constructor, forwarding, and partial application nodes..

The $\bot$ and **free** nodes are the most direct nodes. The $\bot$ node represents a failing computation, and only serves to propagate the failure to the root of the expression. The **free** node represents a free variable.

The ? node represents a choice. We treat choices in a similar way to constructors. We do not immediately choose a branch, but instead defer it until the value is demanded by a case expression. However, this is only an apparent difference. Any expression that is reduced to a choice will immediately demand the value of that choice. This simplifies the execution model, and there does not seem to be a measurable performance loss for delaying the evaluation of choice.

Function and constructor applications are always fully applied. For partial applications, we have the `PART` node. This node contains three things: a function or constructor to apply, a number k represent-

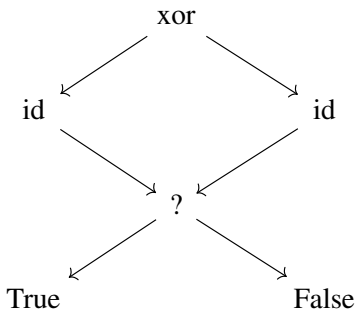
ing the number of arguments the function is missing, and finally the arguments that have already been partially applied. In the implementation, the function is represented by a closure.

Finally, $*(g)$ represents a forwarding, or indirection, node. The notation is supposed to resemble a pointer. Forwarding nodes are necessary for a function that returns one of its parameters. Consider the following example.

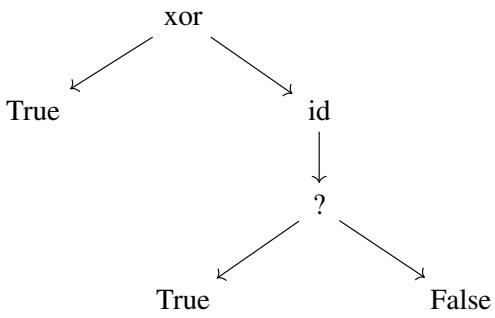
```
id x = x

main = let x = True ? False
       in xor (id x) (id x)
```

The graph representing our expression is:



If we take an approach similar to Peyton Jones [19] and copy the constructor after evaluation, we end up with the following graph.

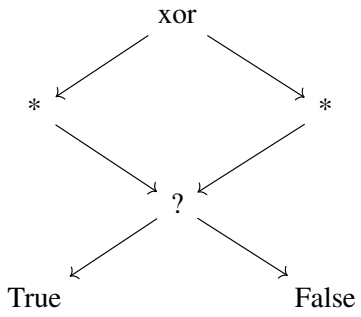


This will certainly cause problems as we try to backtrack, because we need to replace both copies of True. Instead, we solve this problem with the forwarding node $*(g)$. If a Curry function evaluates to a parameter of the function, then we construct a forwarding node to maintain the structure of the graph,

g	$\Rightarrow$	$\bot$	Failed
	$\mid$	**free**	Free variable
	$\mid$	$?(g,g)$	Choice
	$\mid$	$\mathrm{f}(\overline{g})$	Function Application
	$\mid$	$\mathrm{C}(\overline{g})$	Constructor Application
	$\mid$	$*(g)$	Forwarding Node
	$\mid$	$\mathrm{PART}(\mathrm{f},k,\overline{g})$	Partial Application Node

Figure 5: Heap objects represented as a graph.

and prevent any unintended copying. Our example from before evaluates to the following.



6 The Execution Model

To run a Curry program, we evaluate the expression `main` to normal form (or a value). We can accomplish this by successively evaluating `main` to a head normal form, which is a form rooted by a constructor or literal. We then successively evaluate the children of `main` to normal form. If `main` evaluates to a value, then we display it to the user; if it evaluates to $\bot$, we discard the value. In either case, we backtrack and try again. The backtracking scheme is well understood and used in both Pakcs and MCC [14, 24, 16].

In order to implement backtracking, we need to keep track of a backtracking stack. We represent the backtracking stack as a list of frames enclosed in angle brackets.

$S = \langle\rangle$

$S = \langle g_{[?]}, g|S\rangle$

A stack is empty, or it contains two nodes from the heap. The left node is the current value in the heap, and the right node contains a value to replace it with when backtracking. We use the notation $g_?$ to denote that node g came from a choice, and therefore backtracking should stop at this node.

We introduce four relations in Figure 6 for evaluation to normal form, head normal form, backtracking a single step, and backtracking to a choice. The normal form relation evaluates an expression to a value as described above. The graph G and stack S may be changed over the course of evaluation. The evaluation to head normal form is similar, but only evaluates e to an expression rooted by a constructor. Finally, we have two backtracking relations. The first only undoes a single rewrite from the stack. The second one pops rewrites off that stack until we reach a rewrite that came from a choice. The rules for evaluating to normal form and backtracking are given in Figure 7. We use the standard $\Downarrow^n$ notation to refer to the n-fold composition of the $\Downarrow$ relation. Most rules are standard, but the rule for choice may be

$G,S : e \Downarrow_N G,S : v$	evaluation to Normal Form
$G,S : e \Downarrow G,S : v$	evaluation to Head Normal Form
$G,S \Downarrow_B G,S$	Backtracking
$G,S \Downarrow_{B?} G,S$	Backtracking to a choice

Figure 6: evaluation relations
expression e with graph G and stack S evaluates to value v with a possibly modified G and S.

(BT) $$G, \langle x, y | S \rangle \Downarrow_B G[x \mapsto y], S$$

(BT-Choice) $$G, \langle \overline{x,y} \mid l_?, r \mid S \rangle \Downarrow_{B?} G[\overline{x \mapsto y}][l \mapsto r], \langle r, ?(l,r) | S \rangle$$

(Norm-Bot) $$\frac{G,S : e \Downarrow G_1,S_1 : \bot}{G,S : e \Downarrow_N G_1,S_1 : \bot}$$

(Norm-Lit) $$\frac{G,S : e \Downarrow G_1,S_1 : l}{G,S : e \Downarrow_N G_1,S_1 : l}$$

(Norm-Free) $$\frac{G,S : e \Downarrow G_1,S_1 : \mathbf{free}}{G,S : e \Downarrow_N G_1,S_1 : \mathbf{free}}$$

(Norm-Con) $$\frac{G,S : e \Downarrow G_0,S_0 : C(\overline{e}) \qquad \overline{G_i,S_i : e_i \Downarrow_N G_{i+1},S_{i+1} : v_i}}{G,S : e \Downarrow_N G_n,S_n : C(\overline{v})}$$

(Norm-Choice) $$\frac{G,S : e \Downarrow G_1,S_1 : ?(x,y) \qquad G_1[r \mapsto *(x)], \langle r_?, *(y) | S_1 \rangle : x \Downarrow_N G_2,S_2 : v}{G,S : e \Downarrow_N G_2,S_2 : *(v)}$$

Figure 7: backtracking and normalization algorithm.
in (Norm-Choice) r is the root of the expression e

surprising. If an expression is evaluated to a choice, then we choose the left hand side, and evaluate that to normal form. We push the right hand side on the stack so we can backtrack to it later.

While backtracking and evaluation to normal form are typical, evaluation to head normal form requires more explanation. We split the rules up into three parts: basic rules in Figure 8, rules for a `case` in Figure 9, and rules for `apply` in Figure 10.

The basic rules correspond closely to the original FlatCurry semantics with the addition of the stack. The rules are given in Figure 8. The rules for (Bot), (Lit), (Free), and (Con) are already in head normal form, so the evaluation is complete. We also treat ? and * as head normal forms. This is still consistent with the previous semantics because they will both be evaluated by case expressions. The rule for (Let) simply adds each defined variable to the graph. Because all functions are only applied to variables, we treat expression variables the same as graph nodes. The case for (Fun) is very similar to the previous semantics. We replace the function call with the expression graph from the function's definition and continue evaluation.

Finally, the (Var) case is trivial. This seems surprising because the (Nat-VarExp) case was more complicated in the previous semantics. However, the only way an expression could evaluate to a variable that was not the scrutinee of a case expression is if a function returned one of its parameters. In that case,

(Bot) $G,S : \bot \Downarrow G,S : \bot$

(Lit) $G,S : l \Downarrow G,S : l$

(Free) $G,S : \mathbf{free} \Downarrow G,S : \mathbf{free}$

(Con) $G,S : C\ \overline{e} \Downarrow G,S : C(\overline{e})$

(Choice) $G,S : e_1\ ?\ e_2 \Downarrow G,S : ?(e_1,e_2)$

(Fun) $$\frac{f\ \overline{x} = e \qquad G,S : e[\overline{x \mapsto y}] \Downarrow G_1,S_1 : v}{G,S : f\ \overline{y} \Downarrow G_1,S_1 : v}$$

(Let) $$\frac{G[\overline{x \mapsto e}],S : e_1 \Downarrow G_1,S_1 : v}{G,S :\ \mathbf{let}\ \overline{x = e}\ \mathbf{in}\ e_1 \Downarrow G_1,S_1 : v}$$

(Var) $G,S : x \Downarrow G,S : *(x)$

Figure 8: Evaluation of expressions without `case`
We assume all variables from function definitions are fresh.

we need to create a forwarding node for the reasons described above.

More substantial changes start to appear in the case rules. These rules correspond to the while/switch loop in the generated C code for the RICE compiler [23, Chapter 4]. Cases are only applied to variables, so case expressions inspect the variable and evaluate it if necessary. There is one case for each type of heap object we might scrutinize, except for partial applications. Typing rules prevent partial applications from appearing as the scrutinee of a case.

The (Case-Bot) rule is the simplest rule; it only propagates the $\bot$ up. The (Case-Fwd) rule unwraps its argument and tries again. The (Case-Fun) rule evaluates a function, and updates the variable when it returns. This rule pushes x with its old value $f(\overline{y})$ onto the stack, because that rewrite might need to be undone for backtracking. (Case-Choice) will always choose the left-hand side, and push the right-hand side as a non-deterministic rewrite $\langle x_?, *(z) | S \rangle$ onto the backtracking stack. We then update x to be a forwarding node to the left-hand side y. (Case-Lit) and (Case-Con) can select a branch for the case. (Case-Con) has to replace the parameters of the constructor with the arguments. Finally, (Case-LitFree) and (Case-ConFree) handle narrowing steps. The free variable is instantiated to the pattern of the first branch. If the branch is a constructor, then the children are filled with free variables. We use the notation $e[\overline{y_i \mapsto \mathbf{free}_i}]$ to denote replacing each free variable y_i in expression e with the corresponding logic variable that was created in $C(\overline{\mathbf{free}})$. The rest of the patterns are all pushed onto the backtracking stack as rewrites for the free variable.

The final rules are the rules for the `apply` function. We handle apply with the eval-apply method [25]. In fact, the logic features have no bearing on partial application. If we are applying a choice node, then we select the leftmost node and try again. If we are applying a free variable, then we fail.

The remaining possibilities of partial application are split into three cases. If the partial application is under applied, then we simply add the new arguments to the application and move on. If the application is correctly applied, then we evaluate the function with the arguments. Finally, if the application is over

(Case-Bot) $$G[x \mapsto \bot], S : \textbf{case}\ x\ \textbf{of}\ \overline{p \to e} \Downarrow G, S : \bot$$

(Case-Fwd) $$\frac{G, S : \textbf{case}\ y\ \textbf{of}\ \overline{p \to e} \Downarrow G_1, S_1 : v}{G[x \mapsto *(y)], S : \textbf{case}\ x\ \textbf{of}\ \overline{p \to e} \Downarrow G_1, S_1 : v}$$

(Case-Fun) $$\frac{G, S : f\ \overline{y} \Downarrow G_1, S_1 : v_x \quad G_1[x \mapsto v_x], \langle x, f(\overline{y}) | S_1 \rangle : \textbf{case}\ v_x\ \textbf{of}\ \overline{p \to e} \Downarrow G_2, S_2 : v}{G[x \mapsto f(\overline{y})], S : \textbf{case}\ x\ \textbf{of}\ \overline{p \to e} \Downarrow G_2, S_2 : v}$$

(Case-Choice) $$\frac{G[x \mapsto *(y)], \langle x_?, *(z) | S \rangle : \textbf{case}\ y\ \textbf{of}\ \overline{p \to e} \Downarrow G_1, S_1 : v}{G[x \mapsto ?(y,z)], S : \textbf{case}\ x\ \textbf{of}\ \overline{p \to e} \Downarrow G_1, S_1 : v}$$

(Case-Lit) $$\frac{G, S : e_i \Downarrow G_1, S_1 : v}{G[x \mapsto l_i], S : \textbf{case}\ x\ \textbf{of}\ \overline{l \to e} \Downarrow G_1, S_1 : v}$$

(Case-LitFree) $$\frac{G[x \mapsto l_1], S_L : e_1 \Downarrow G_1, S_1 : v}{G[x \mapsto \textbf{free}], S : \textbf{case}\ x\ \textbf{of}\ \overline{l \to e} \Downarrow G_1, S_1 : v}$$

(Case-Con) $$\frac{G, S : e_i[\overline{y \mapsto z}] \Downarrow G_1, S_1 : v}{G[x \mapsto C_i(\overline{z})], S : \textbf{case}\ x\ \textbf{of}\ \overline{C\ \overline{y} \to e} \Downarrow G_1, S_1 : v}$$

(Case-ConFree) $$\frac{G[x \mapsto C_1(\overline{\textbf{free}})], S_C : e_1[\overline{y_i \mapsto \textbf{free}_i}] \Downarrow G_1, S_1 : v}{G[x \mapsto \textbf{free}], S : \textbf{case}\ x\ \textbf{of}\ \overline{C\ \overline{y} \to e} \Downarrow G_1, S_1 : v}$$

Figure 9: rules for Case expressions.
In Case-LitFree $S_L = \langle x, l_2 | \ldots | x, l_n | x, \textbf{free} | S_1 \rangle$
In Case-ConFree $S_C = \langle x, C_2(\overline{\textbf{free}}) | \ldots | x, C_n(\overline{\textbf{free}}) | x, \textbf{free} | S_1 \rangle$

$$\text{(Apply-Free)} \quad G[x \mapsto \textbf{free}], S : \text{apply } x\ \overline{e} \Downarrow G, S : \bot$$

$$\text{(Apply-Choice)} \quad \frac{G[x \mapsto *(y)], \langle x_?, *(z) | S \rangle : \text{apply } y\ \overline{e} \Downarrow G_1, S_1 : v}{G[x \mapsto ?(y,z)], S : \text{apply } x\ \overline{e} \Downarrow G_1, S_1 : v}$$

$$\text{(Apply-Under)} \quad \frac{|\overline{e}| = n < k}{G_x, S : \text{apply } x\ \overline{e} \Downarrow G, S : \text{PART}(f, k-n, \overline{ye})}$$

$$\text{(Apply-Full)} \quad \frac{G, S : f\ \overline{y}\ \overline{e_k} \Downarrow G_1, S_1 : v}{G_x, S : \text{apply } x\ \overline{e_k} \Downarrow G_1, S_1 : v}$$

$$\text{(Apply-Over)} \quad \frac{G, S : f\ \overline{y}\ \overline{e_k} \Downarrow G_1, S_1 : v_1 \qquad G_1, S_1 : \text{apply } v_1\ \overline{e_{k+1}} \Downarrow G_2, S_2 : v_2}{G_x, S : \text{apply } x\ \overline{e_k e_{k+1}} \Downarrow G_2, S_2 : v_2}$$

Figure 10: apply rules.
In all 3 rules $G_x = G[x \mapsto \text{PART}(f, k, \overline{y})]$
In (apply-over) $\overline{e_k} = e_1 \ldots e_k$
In (apply-over) $\overline{e_{k+1}} = e_{k+1} \ldots e_n$

applied, then it must evaluate to a `PART`; we take the first few arguments, evaluate to the `PART`, and supply the final arguments.

7 Correspondence to Generated Code

The purpose of this semantics is to model the execution of programs compiled with RICE. In order to justify that the semantics really does correspond with the compiled code, we give a small example of compiled RICE code for the not function defined below

```
not x = case x of
            True -> False
            False -> True
```

In the RICE compiler Curry expressions are compiled C code. The graph nodes from the semantics are represented as Node objects, which correspond to closures in a traditional functional language. Its definition is given in Figure 11. Each Node has 3 fields. The missing field is used for partial application, the symbol field contains information about the Node such as it's name and arity and tag describing what node it is, as well as a pointer to code to reduce the node to head normal form. Finally, each node has an array of 4 children. If a node has arity greater than 4, the final slot is a pointer to an array containing the rest of the children.

The code for reducing the expression not e for some expression e is given in Figure 12. We continue to loop until we reduce to a head normal form. The branches in the case corresponding to the different Case rules in Figure 9. The rules for FAIL, True, and False just set the symbol, and remove the child and return. The forward tag sets the scrutinee to be its first child and retries. The choice tag makes sets the scrutinee to be one of its children and pushes in on the stack. The details are elided here. Finally, The function rule reduces the scrutinee and pushes it on the backtracking stack which we call bt_stack .

```
typedef struct Node {
    const unsigned char tag;
    int missing;
    Symbol* symbol;
    field children[4];
} Node;
```

Figure 11: Definition for a Node object

```
void not_hnf(field root) {
  Node* scrutinee = root->children[0];
  while(true) {
    switch(scrutinee->tag) {
      case FAIL_TAG:
        root->scrutinee = FAIL_symbol;
        root->children[0] = NULL;
        return;
      case FORWARD_TAG:
        scrutinee = scrutinee->children[0];
        break;
      case CHOICE_TAG:
        choose(scrutinee);
        break;
      case FUNCTION_TAG:
        scrutinee->symbol->hnf(srutinee);
        push(bt_stack, scrutinee, false);
        break;
      case True_TAG:
        root->symbol = False_Symbol;
        root->children[0] = NULL;
        return;
      case False_TAG:
        root->symbol = True_Symbol;
        root->children[0] = NULL;
        return;
    }
  }
}
```

Figure 12: Code for reducing a *not* node to head normal form.

8 Correctness

With the semantics now established, we need to show that they actually implement Curry. We do this via comparison to the original semantics [2]. However, the original semantics were non-deterministic, and we cannot hope to match them. Because we are not using a fair evaluation strategy, there will be answers that we may not produce in a finite amount of time.

Nevertheless, we can still show that we agree with the original semantics in restricted cases. Specifically, we show that for terminating programs, we produce the same answers as the original semantics, which is not surprising. If we remove the stack from our semantics, then we match the original fairly closely.

We start by proving that the backtracking operation does backtrack as we claim. Specifically, we show that if we have a graph G and expression e that we evaluate to the new graph G' and v, then there is some number of backtracking steps that will restore the original graph.

Theorem 1. *For any expression graph G, stack S, and expression e if* $G,S : e \Downarrow_N G',S' : v$, *then there exists some n where* $G',S' \Downarrow_B^n G \cup G_u, S$ *where* G_u *is a set of unreachable bindings created after e was evaluated.*

Proof. The proof is by structural induction on the derivation of e. The only rules that alter G are (Let), (Case-Choice), (Case-Fun). (Case-LitFree), (Case-ConFree), (Norm-Choice), and (Apply-Choice). All other cases are trivial because they do not modify the stack or the graph. The (Let) rule can only add new bindings to the graph, so these new bindings may go in G_u.

(Case-Fun): If $G[x \mapsto f(\overline{y})]$, then there are two evaluations that take place: $G,S : f(\overline{y}) \Downarrow G_1,S_1 : v_x$ and $G[x \mapsto v_x], \langle x, f(\overline{y}) | S_1 \rangle$: **case** v_x **of** $\overline{p \to e} \Downarrow G_2,S_2 : v$. By our inductive hypotheses $G_2,S_2 \Downarrow_B^n G_1[x \mapsto v_x] \cup G_{u2}, \langle x, f(\overline{y}) | S_1 \rangle$, so $G_2,S_2 \Downarrow_B^{n+1} G_1[x \mapsto f(\overline{y})] \cup G_{u2}, S_1$. Again, by our induction hypothesis $G_1,S_1 \Downarrow_B^m G \cup G_{u1}, S$. Therefore, $G_2,S_2 \Downarrow_B^{m+n+1} G \cup (G_{u1} \cup G_{u2}), S$.

The cases for (Case-Choice), (Case-LitFree), (Case-ConFree), (Norm-Choice), and (Apply-Choice) are all similar, but there is one slight alteration. We use (Case-Choice) as an example. If $G[x \mapsto ?(y,z)]$, then there is only one derivation. $G[x \mapsto *(y)], \langle x_?, *(z) | S \rangle$: **case** y **of** $\overline{p \to e} \Downarrow G_1,S_1 : v$ By the induction hypothesis $G_1,S_1 \Downarrow_B^n G[x \mapsto *(y)] \cup G_n, \langle x_?, *(z) | S \rangle$. Our next backtracking step will replace y with z and add something new to the stack. $G_1,S_1 \Downarrow_B^{n+1} G[x \mapsto *(z)] \cup G_n, \langle x, ?(y,z) | S \rangle$. This is fine because the next backtracking step will restore the stack. $G_1,S_1 \Downarrow_B^{n+2} G \cup G_n, S$. This completes the proof. □

Next, we show that for any graph G, stack S, and expression e, if we evaluate using our semantics, then we produce the same value as the natural semantics [2] assuming all choices choose the left hand side. We use $\Downarrow_C$ as the evaluation relation from the natural semantics. We recall the rules for the natural semantics in Figure 2. Because there is only a relation for head normal forms, and not normal forms, we restrict ourselves to evaluations that terminate in a constructor or literal.

Theorem 2. *If* $G,S : e \Downarrow G',S' : v$, *Where v is a constructor or a literal, then there is a heap* Γ *that corresponds to G, and a heap* Γ' *corresponding to* G' *such that* $\Gamma : e \Downarrow_C \Gamma' : v'$ *where* v' *is the same as v with the forward nodes contracted.*

Proof. We prove this by constructing a transformation on derivations in our semantics to a derivation in the natural semantics. Because the natural semantics is not formulated for higher order expressions, we will assume all expressions are first order and all applications are fully applied.

We create a mapping $\Leftrightarrow$ which maps evaluation rules from our semantics to the natural semantics. The full mapping can be found in Figure 13. By our assumption, (Bot) or (Case-Bot) can never appear in

the evaluation. If they did, then the $\perp$ would propagate to the root of the expression. The cases for (Lit), (Con), (Free), (Fun), and (Let) are straightforward. We elide the stack in all of these mappings because it is not relevant to the proof.

The only two non-case rules that do not directly correspond are (Choice) and (Var). By our assumption that v is a constructor or a literal, we know that these rules must appear in the context of a case expression. Because the scrutinee of all case statements is a variable, all of our case rules will correspond to multiple rules in the natural semantics. Specifically, every scrutinee that is not in head normal form will have a (Case-Fun) rule to evaluate it. We can assume that there may be (Case-Fwd) rules before any of the case rules are applied. This does not affect the result because the forwarding nodes will disappear after contraction. We show the case for a function application that evaluates to a literal, but the case for a function evaluating to a constructor is identical.

In the case of (Choice), the correspondence is not immediately clear. The evaluation seems very different because we treat choice as a head normal form and the natural semantics does not. However, because all choices must be evaluated in a case, the next step in the evaluation is to select a branch for the choice.

Finally, we will consider the narrowing step. This is similar to choice in that free variables are normal forms, but it is an easier correspondence because free variables are normal forms in the natural semantics as well. In the natural semantics, if an expression e evaluates to a variable x, then it must be the case that $\Gamma[x \mapsto x]$ and x is a free variable. We show the case for case expressions with literal branches, but the constructor case is identical. Because this covers every rule, $\Leftrightarrow$ is a correspondence between our semantics and the natural semantics. □

These two theorems justify the correctness of our execution model. If we have a terminating expression e, and e evaluates to a value v in the natural semantics, then it will eventually evaluate to v in our semantics.

9 Related Work and Conclusion

This work was built on the work of Hanus et al. [2], and Braßel [11]. Our execution model follows the execution model of Pakcs [14], with improvements for performance. There are a number of different semantics for Curry including CRWL [28], and rewriting [16]. We elected to go with the natural semantics because it closely resembles the implementation of the RICE compiler. Other execution models have been described for MCC [24] and KiCS2 [12]. We cannot directly use the work on KiCS2 because it uses pull-tabbing rather than backtracking. The execution model in MCC was different enough that we did not feel it was useful to build on it.

Another alternative would be the semantics given for the Verse Calculus [9]. While we think it would be an interesting idea to compile Curry to VC and see how the performance compares, we still have many questions about implementation details.

In future work, we would like to show the correctness of some of the optimizations to the execution model found in the RICE compiler [23]. These include fast backtracking [22] and case shortcutting [23]. We would also like to show a correspondence with the denotational semantics given by Mehner et al. [27] to use the free theorems to justify some tricky compiler transformations.

We have presented the execution model for RICE. We extended the natural semantics by making it deterministic and adding a stack. We justified our execution model by showing that expressions evaluate to the same values as the natural semantics. We believe that this execution model is simple enough to be understandable, but detailed enough to be useful.

(Lit) $G : l \Downarrow G : l$

$\Leftrightarrow$

(Nat-Val) $\Gamma : l \Downarrow \Gamma : l$

(Con) $G : C\ \overline{e} \Downarrow G : C(\overline{e})$

$\Leftrightarrow$

(Nat-Val) $\Gamma : C\ \overline{e} \Downarrow \Gamma : C\ \overline{e}$

(Fun)
$$\frac{f\ \overline{x} = e \qquad G : e[\overline{x \mapsto y}] \Downarrow G_1 : v}{G : f\ \overline{y} \Downarrow G_1 : v}$$

$\Leftrightarrow$

(Nat-Fun)
$$\frac{\Gamma : \rho(e) \Downarrow \Delta : v}{\Gamma : f\ \overline{y} \Downarrow \Delta : v} \text{ where } f\ \overline{x} \in P \text{ and } \rho(y_n) = x_n$$

(Let)
$$\frac{G[\overline{x \mapsto e}] : e_1 \Downarrow G_1 : v}{G : \textbf{ let } \overline{x = e} \textbf{ in } e_1 \Downarrow G_1 : v}$$

$\Leftrightarrow$

(Nat-Let)
$$\frac{\Gamma[\overline{y_k \mapsto \rho(e_k)}] : e \Downarrow \Delta : v}{\Gamma : \textbf{let } \overline{x_k = e_k} \textbf{ in } e \Downarrow \Delta : v} \text{ where } \rho(x_k) = y_k$$

(Case-Fun-Lit)
$$\frac{G : f\ \overline{y} \Downarrow G_1 : l_i \qquad \dfrac{G_1 : e_i \Downarrow G_2 : v}{G_1[x \mapsto l_i] : \textbf{case } v_x \textbf{ of } \overline{l \to e} \Downarrow G_2 : v}}{G[x \mapsto f(\overline{y})] : \textbf{case } x \textbf{ of } \overline{l \to e} \Downarrow G_2 : v}$$

$\Leftrightarrow$

(Nat-Select)
$$\frac{\dfrac{\Gamma : \rho(e) \Downarrow \Delta : l_i}{\Gamma : f\ \overline{y} \Downarrow \Delta : l_i} \qquad \Delta : e_i \Downarrow \Phi : v}{\Gamma[x \mapsto f\ \overline{y}] : \textbf{case } x \textbf{ of } \overline{l_i \to e_i} \Downarrow \Phi : v} \text{ where } f\ \overline{x} \in P \text{ and } \rho(y_n) = x_n$$

(Case-Choice)
$$\frac{G : f\ \overline{y} \Downarrow G_1 : ?(y,z) \qquad \dfrac{\dfrac{\dfrac{G_1 : e \Downarrow G_2 : p_i \qquad G_2 : e_i \Downarrow G_3 : v}{G_1[y \mapsto e] : \textbf{case } y \textbf{ of } \overline{p \to e} \Downarrow G_3 : v}}{G_1[x \mapsto *(y)] : \textbf{case } x \textbf{ of } \overline{p \to e} \Downarrow G_3 : v}}{G_1[x \mapsto ?(y,z)] : \textbf{case } x \textbf{ of } \overline{p \to e} \Downarrow G_3 : v}}{G_1[x \mapsto f(\overline{y})] : \textbf{case } x \textbf{ of } \overline{l \to e} \Downarrow G_3 : v}$$

$\Leftrightarrow$

(Nat-Or)
$$\frac{\dfrac{\dfrac{\dfrac{\Gamma : y \Downarrow \Delta : p_i}{\Gamma : y \ or\ z \Downarrow \Delta : p_i}}{\vdots}}{\dfrac{\Gamma : \rho(e) \Downarrow \Delta : p_i}{\Gamma : f\ \overline{y} \Downarrow \Delta : p_i}} \qquad \Delta : e_i \Downarrow \Phi : v}{\Gamma[x \mapsto f\ \overline{y}] : \textbf{case } x \textbf{ of } \overline{l_i \to e_i} \Downarrow \Phi : v}$$

(Case-LitFree)
$$\frac{G[x \mapsto l_1] : e_1 \Downarrow G_1 : v}{G[x \mapsto \textbf{free}] : \textbf{case } x \textbf{ of } \overline{l \to e} \Downarrow G_1 : v}$$

$\Leftrightarrow$

(Nat-Guess)
$$\frac{\Gamma : e \Downarrow \Delta : x \qquad \Delta[x \mapsto l_1] : e_1 \Downarrow \Theta : v}{\Gamma[x \mapsto e] : \textbf{case } x \textbf{ of } \overline{l \to e} \Downarrow \Theta : v}$$

Figure 13: The mapping $\Leftrightarrow$

References

[1] E. Albert, M. Hanus, F. Huch, J. Oliver & G. Vidal (2002): *A Deterministic Operational Semantics for Functional Logic Programs*, p. 207. Available at `https://www.programmazionelogica.it/wp-content/uploads/2002/09/agp02_207.pdf`.

[2] E. Albert, M. Hanus, F. Huch, J. Oliver & G. Vidal (2005): *Operational semantics for declarative multi-paradigm languages*. *Journal of Symbolic Computation* 40(1), pp. 795–829, doi:10.1016/j.jsc.2004.01.001.

[3] S. Antoy (1992): *Definitional Trees*. In H. Kirchner & G. Levi, editors: *Algebraic and Logic Programming, Third International Conference, Volterra, Italy, September 2-4, 1992, Proceedings*, *Lecture Notes in Computer Science* 632, Springer, pp. 143–157, doi:10.1007/BFB0013825.

[4] S. Antoy (1997): *Optimal Non-deterministic Functional Logic Computations*. In M. Hanus, J. Heering & K. Meinke, editors: *Algebraic and Logic Programming, 6th International Joint Conference, ALP '97 - HOA '97, Southampton, UK, September 3-5, 1997, Proceedings*, *Lecture Notes in Computer Science* 1298, Springer, pp. 16–30, doi:10.1007/BFB0027000.

[5] S. Antoy, R. Echahed & M. Hanus (2000): *A needed narrowing strategy*. *J. ACM* 47(4), pp. 776–822, doi:10.1145/347476.347484.

[6] S. Antoy, M. Hanus, A. Jost & S. Libby (2019): *ICurry* 12057, pp. 286–307. doi:10.1007/978-3-030-46714-2_18.

[7] S. Antoy & A. Jost (2016): *A New Functional-Logic Compiler for Curry: Sprite* 10184, pp. 97–113. doi:10.1007/978-3-319-63139-4_6.

[8] A. W. Appel (2006): *Compiling with Continuations (corr. version)*. Cambridge University Press.

[9] L. Augustsson, J. Breitner, K. Claessen, R. Jhala, S. Peyton Jones, O. Shivers, G. L. Steele Jr. & T. Sweeney (2023): *The Verse Calculus: A Core Calculus for Deterministic Functional Logic Programming*. *Proceedings of the ACM on Programming Languages* 7(ICFP), pp. 417–447, doi:10.1145/3607845.

[10] J. Böhm, M. Hanus & F. Teegen (2021): *From Non-determinism to Goroutines: A Fair Implementation of Curry in Go*. In N.ò Veltri, N. Benton & S. Ghilezan, editors: *PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021*, ACM, pp. 16:1–16:15, doi:10.1145/3479394.3479411.

[11] B. Braßel (2010): *Implementing Functional Logic Programs by Translation into Purely Functional Programs*. Ph.D. thesis, University of Kiel. Available at `http://eldiss.uni-kiel.de/macau/receive/dissertation_diss_00007056`.

[12] B. Braßel, M. Hanus, B. Peemöller & F. Reck (2011): *KiCS2: A New Compiler from Curry to Haskell*. In: H. Kuchen, editor: *Functional and Constraint Logic Programming, 20th International Workshop, WFLP 2011, Odense, Denmark, July 19, 2011, Proceedings*, *Lecture Notes in Computer Science* 6816, Springer, pp. 1–18, doi:10.1007/978-3-642-22531-4_1.

[13] R. Echahed & J. C. Janodet (1997): *On constructor-based graph rewriting systems*. Technical Report 985-I, IMAG. Available at ftp://ftp.imag.fr/pub/labo-LEIBNIZ/OLD-archives/PMP/c-graph-rewriting.ps.gz.

[14] M. Hanus (ed.) (March 04, 2017): *PAKCS 1.14.3: The Portland Aachen Kiel Curry System*. Available at `http://www.informatik.uni-kiel.de/~pakcs`.

[15] C. Flanagan, A. Sabry, B. F. Duba & M. Felleisen (1993): *The Essence of Compiling with Continuations*. In R. Cartwright, editor: *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation (PLDI), Albuquerque, New Mexico, USA, June 23-25, 1993*, ACM, pp. 237–247, doi:10.1145/155090.155113.

[16] M. Hanus (2013): *Functional Logic Programming: From Theory to Curry*, pp. 123–168. *Lecture Notes in Computer Science* 7797, Springer, doi:10.1007/978-3-642-37651-1_6.

[17] H. Hußmann (1988): *Nondeterministic Algebraic Specifications and Nonconfluent Term Rewriting*. In J. Grabowski, P. Lescanne & W. Wechler, editors: *Algebraic and Logic Programming, International Work-*

shop, Gaussig, GDR, November 14-18, 1988, Proceedings, *Lecture Notes in Computer Science* 343, Springer, pp. 31–40, doi:10.1007/3-540-50667-5_56.

[18] T. Johnsson (1985): *Lambda Lifting: Treansforming Programs to Recursive Equations*. In J. Jouannaud, editor: *Functional Programming Languages and Computer Architecture, FPCA 1985, Nancy, France, September 16-19, 1985, Proceedings*, *Lecture Notes in Computer Science* 201, Springer, pp. 190–203, doi:10.1007/3-540-15975-4_37.

[19] S. L. Peyton Jones (1987): *The Implementation of Functional Programming Languages*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[20] S. L. Peyton Jones & J. Salkild (1989): *The Spineless Tagless G-Machine*. In J. E. Stoy, editor: *Proceedings of the fourth international conference on Functional programming languages and computer architecture, FPCA 1989, London, UK, September 11-13, 1989*, ACM, pp. 184–201, doi:10.1145/99370.99385.

[21] J. Launchbury (1993): *A Natural Semantics for Lazy Evaluation*. In: *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '93, Association for Computing Machinery, New York, NY, USA, p. 144–154, doi:10.1145/158511.158618.

[22] S. Libby (2023): *RICE: An Optimizing Curry Compiler*. In M. Hanus & D. Inclezan, editors: *Practical Aspects of Declarative Languages - 25th International Symposium, PADL 2023, Boston, MA, USA, January 16-17, 2023, Proceedings*, *Lecture Notes in Computer Science* 13880, Springer, pp. 3–19, doi:10.1007/978-3-031-24841-2_1.

[23] S. Libby (June 21, 2022): *Making Curry with Rice: An Optimizing Curry Compiler*. Ph.D. thesis, Portland State University, doi:10.15760/etd.7964. Avalible at https://github.com/slibby05/rice.

[24] W. Lux & H. Kuchen (1999): *An Efficient Abstract Machine for Curry*. In K. Beiersdörfer, G. Engels & W. Schäfer, editors: *Informatik '99 - Informatik überwindet Grenzen, 29. Jahrestagung der Gesellschaft für Informatik, Paderborn, 5.-9. Oktober 1999*, Informatik Aktuell, Springer, pp. 390–399, doi:10.1007/978-3-662-01069-3_58.

[25] S. Marlow & S. L. Peyton Jones (2004): *Making a fast curry: push/enter vs. eval/apply for higher-order languages*. In C. Okasaki & K. Fisher, editors: *Proceedings of the Ninth ACM SIGPLAN International Conference on Functional Programming, ICFP 2004, Snow Bird, UT, USA, September 19-21, 2004*, ACM, pp. 4–15, doi:10.1145/1016850.1016856.

[26] R. Mayr & T. Nipkow (1998): *Higher-Order Rewrite Systems and Their Confluence*. *Theoretical Computer Science* 192(1), pp. 3–29, doi:10.1016/S0304-3975(97)00143-6.

[27] S. Mehner, D. Seidel, L. Straßburger & J. Voigtländer (2014): *Parametricity and Proving Free Theorems for Functional-Logic Languages*. In O. Chitil, A. King & O. Danvy, editors: *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming, Kent, Canterbury, United Kingdom, September 8-10, 2014*, ACM, pp. 19–30, doi:10.1145/2643135.2643147.

[28] J. C. González Moreno, M. T. Hortalá-González, F. J. López-Fraguas & M. Rodríguez-Artalejo (1996): *A Rewriting Logic for Declarative Programming*. In H. R. Nielson, editor: *Programming Languages and Systems - ESOP'96, 6th European Symposium on Programming, Linköping, Sweden, April 22-24, 1996, Proceedings*, *Lecture Notes in Computer Science* 1058, Springer, pp. 156–172, doi:10.1007/3-540-61055-3_35.

Paraconsistent Relations as a Variant of Kleene Algebras *

Juliana Cunha
CIDMA, Dep. Mathematics, Aveiro University
Aveiro, Portugal
INESC TEC & Dep. Informatics, Minho University
Braga, Portugal
juliana.cunha@ua.pt

Alexandre Madeira
CIDMA, Dep. Mathematics, Aveiro University
Aveiro, Portugal
madeira@ua.pt

Luís S. Barbosa
INESC TEC & Dep. Informatics, Minho University
Braga, Portugal
lsb@di.uminho.pt

Kleene algebras (KA) and Kleene algebras with tests (KAT) provide an algebraic framework to capture the behavior of conventional programming constructs. This paper explores a broader understanding of these structures, in order to enable the expression of programs and tests yielding vague or inconsistent outcomes. Within this context, we introduce the concept of a paraconsistent Kleene Algebra with tests (PKAT), capable of capturing vague and contradictory computations. Finally, to establish the semantics of such a structure, we introduce two algebras, $\mathbf{Set}_{\mathbf{P}}(\mathcal{T})$ and $\mathbf{Rel}_{\mathbf{P}}(\mathcal{K},\mathcal{T})$, parametric on a class of twisted structures $\mathcal{K}$ and $\mathcal{T}$. We believe this sort of structures, for their huge flexibility, have an interesting application potential.

1 Introduction

In his seminal work [26], Stephen Kleene described finite deterministic automata along with a specification language: regular expressions. Kleene's paper left open the question of whether a finite, sound, and complete axiomatization of the equivalence of regular expressions existed, which would provide an algebraic framework for describing regular languages. This question has been explored by many researchers. In 1964, Redko [37] proved that no finite set of equational axioms could fully characterize the algebra of regular expressions, and in 1966, Salomaa [38] provided two complete axiomatizations of this algebra. Conway's comprehensive 1971 work [13] presented a detailed overview of results related to regular expressions and their axiomatizations. In [28] Kozen showed that Salomaa's axiomatization is non-algebraic,i.e., unsound under substitution of alphabet symbols by arbitrary regular expressions. He then presented an algebraic axiomatization: Kleene algebras (KA). Later in [29], Kozen also introduced a variant of KA called Kleene Algebras with Tests (KAT). The addition of tests to KA was prompted by the need to express conventional constructs such as conditionals and **while** loops. As a result, KAT is specifically designed for equational reasoning about these constructs. For any proposition α it is possible to form a test α? that acts as a guard: it succeeds with no side effects in states satisfying α, and fails or aborts otherwise[1].

*This work was financed by PRR - Plano de Recuperação e Resiliência under the Next Generation EU from the European Union within Project Agenda ILLIANCE C644919832-00000035 - Project n 46, as well as by National Funds through FCT - Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) within the project IBEX, with reference PTDC/CCI-COM/4280/2021 (DOI 10.54499/PTDC/CCI-COM/4280/2021)

[1]Notice that throughout this paper, distinct symbols will be used for programs and propositions. Therefore, we often omit the ? symbol and simply write α to denote a test.

Cynthia Kop and Helida Salles Santos (Eds); 19th Workshop on
Logical and Semantic Frameworks with Applications (LSFA'24)
EPTCS 421, 2025, pp. 130–147, doi:10.4204/EPTCS.421.8

While historically prominent in automata theory and formal languages [32], KA and their variants have found applications across various domains, including relational algebra [39], program semantics and logics [33], compiler optimization [30], and algorithm design and analysis [27]. For a more comprehensive read of its applications, the reader is referred to [4].

KAT are suitable to reason about imperative programs since these programs can be thought of as sequences of discrete steps, each related to an atomic transition in a standard automaton. Traditionally, these programs operate within a bivalent truth space, where assertions have Boolean outcomes. While this framework has proven to suit a huge range of computer science applications, its inherent simplicity and rigidity may fall short in capturing some intricacies present in real-world scenarios.

Actually, it is not uncommon in software engineering to face scenarios in which vague, or weakly consistent, or even contradictory information is present. Often such characteristics cannot be abstracted away or swept under the carpet. A typical example the authors are currently facing, and which forms one of our motivations for this work, concerns repositories of medical images in a particular domain, which are labeled by several medical judgments from different expert teams, which often are in partial contradiction. Another example arises in the analysis of implementations of quantum circuits in current NISQ (Noisy Intermediate-Scale Quantum) technology [34], where conflicting decoherence levels in the quantum memory have to be taken into account.

Therefore, the introduction of algebraic structures to model computations becomes necessary when the behavior of these computations does not conform to a simple bivalent outcome. Instead, it may involve weighted outcomes from a richer domain, potentially lacking consistency. In this context, vagueness captures the lack of information, while paraconsistency, a well-established designation in logic, expresses the excess of information arising from contradictory judgments.

From a technical point of view, paraconsistency refers to a property inherent to a consequence relation. A logic is said to be "paraconsistent" if and only if its logical consequence relation, whether semantic or proof-theoretic, does not lead to explosion [35]. In logic, the term "explosion" refers to the principle of *ex falso quodlibet*, meaning "from contradiction, anything follows". This principle is the basis for the law of non-contradiction in classical logic. It asserts that from contradictory premises, any proposition can be derived, thus leading to triviality (where anything can be proved true). Consequently, paraconsistent logics set themselves apart from classical ones by their capacity to handle inconsistent information without "exploding" into absurdity. Initially developed in Latin America during the 1950s and 1960s, notably through the influential contributions of F. Asenjo and Newton da Costa, paraconsistent logic quickly garnered attention within the logic and computer science communities. Its original focus on mathematical applications has since expanded, as evidenced by recent literature emphasizing the engineering potential of paraconsistency [3]. Relevant applications are documented in several fields, including deontic logic [14], data network monitoring [21], robotics [1], quantum mechanics [12] and quantum information theory [2],

Since their introduction numerous formalizations of paraconsistent logic have emerged [15, 22]. This paper takes a specific standpoint: the notion of paraconsistent transition systems (PLTS) introduced in [17] and later used to reason about decoherence in quantum circuits as documented in [5]. Informally, in these systems, each transition is assigned a pair of weights: a *positive* weight representing the evidence supporting the transition's occurrence, and a *negative* weight representing the evidence against it.

This paper aims at developing an algebraic counterpart to our research on PLTS, already documented in a number of references [5, 16, 17, 19, 20]. As in previous works, we adopt a similar approach to that in [8], focusing on a particular class of residuated lattices over a set A of possible truth values. In this setting, both the positive and negative weights are taken from the set A.

Furthermore, to establish the groundwork for the sequel, we work with the concept of a *twisted-*

structure, originally proposed by Kalman [25]. This structure arises from the direct product of a lattice $\mathbf{L}$ with its order-dual $\mathbf{L}^{\partial}$ and serves as a key tool for jointly computing positive and negative weights in PLTS, represented by pairs in $L \times L^{\partial}$. Additionally, the twisted-structure naturally carries a De Morgan involution, which we denote by $/\!/$ and interpret as a form of "negation".

The main contribution of this paper lies in the proposal of an extension of KAT to a paraconsistent framework able to reason about uncertain or inconsistent computations. This allows computations to yield outcomes graded by two weights: one indicating evidence for execution and the other evidence for failure. Note that a similar motivation can be found in [23] where a graded variant of KAT is introduced. That work, however, only captures forms of uncertainty as usual in fuzzy logic. Finally, we present two examples of PKAT, paraconsistent sets and relations, which are parametric on arbitrary twisted structures resulting from the direct product of the relevant lattices. These examples serve to illustrate how PKAT handles computations with a paraconsistent reasoning.

Paper structure. Subsection 1.1 recalls the definition of KA and KAT. Section 2 revisits the definition of PLTS parametric on a class of residuated lattices [17] and establishes some new properties that will prove useful in the sequel. Section 3 introduces a variant of KAT for a paraconsistent context (PKAT) where programs and tests accommodate inconsistencies and vagueness. Additionally, in Section 3 we present the details of two new algebraic structures that form a PKAT: paraconsistent sets $\mathbf{Set_P}(\mathcal{T})$ and paraconsistent relations $\mathbf{Rel_P}(\mathcal{K},\mathcal{T})$, which are parametric over fixed twisted structures $\mathcal{K}$ and $\mathcal{T}$, respectively. Finally, Section 4 concludes and points out a number of topics for future research.

1.1 Preliminaries

Definition 1. *[29] A Kleene algebra (KA) is an algebraic structure* $(K,+,\cdot,^{\star},0,1)$ *satisfying the axioms* (1)-(13) *below. The order of precedence of the operators is* $^{\star} > \cdot > +$. *Thus,* $p+q\cdot r^{\star}$ *should be parsed as* $p+(q\cdot(r^{\star}))$.

$$p+(q+r) = (p+q)+r \tag{1}$$
$$p+q = q+p \tag{2}$$
$$p+0 = p \tag{3}$$
$$p+p = p \tag{4}$$
$$p\cdot(q\cdot r) = (p\cdot q)\cdot r \tag{5}$$
$$1\cdot p = p\cdot 1 = p \tag{6}$$
$$p\cdot(q+r) = p\cdot q+p\cdot r \tag{7}$$
$$(p+q)\cdot r = p\cdot r+q\cdot r \tag{8}$$
$$0\cdot p = p\cdot 0 = 0 \tag{9}$$
$$1+p\cdot p^{\star} = p^{\star} \tag{10}$$
$$1+p^{\star}\cdot p = p^{\star} \tag{11}$$
$$p\cdot r \leq r \longrightarrow p^{\star}\cdot r \leq r \tag{12}$$
$$r\cdot p \leq r \longrightarrow r\cdot p^{\star} \leq r \tag{13}$$

where $\leq$ *refers to the natural partial order on K, that is,* $p \leq q$ *if and only if* $p+q = q$.

Axioms (1)-(9) establish $(K,+,\cdot,^{\star},0,1)$ as an idempotent semiring, while axioms (10)-(13) say that * is like the reflexive transitive closure on binary relations [24].

Definition 2. *[24] A* Kleene algebra with tests *(KAT) is a two-sorted algebra*

$$(K,B,+,\cdot,^{\star},^{-},0,1)$$

such that $(K,+,\cdot,^{\star},0,1)$ *is a Kleene algebra,* $(B,+,\cdot,^{-},0,1)$ *is a Boolean algebra and* $B \subseteq K$.

The unary operator $^{-}$ is defined only on B which elements are called tests. We reserve the letters p, q, r, s for arbitrary elements of K and α, β, γ for tests. In summary, a KAT satisfies axioms (1)-(13) and the following for any tests:

$$\alpha + (\beta \cdot \gamma) = (\alpha + \beta) \cdot (\alpha + \gamma) \quad (14) \qquad \overline{\overline{\alpha}} = \alpha \quad (18)$$
$$(\alpha \cdot \beta) + \gamma = (\alpha + \gamma) \cdot (\beta + \gamma) \quad (15) \qquad \alpha + 1 = 1 \quad (19)$$
$$\alpha \cdot \beta = \beta \cdot \alpha \quad (16) \qquad \alpha \cdot \overline{\alpha} = 0 \quad (20)$$
$$\alpha \cdot \alpha = \alpha \quad (17) \qquad \alpha + \overline{\alpha} = 1 \quad (21)$$

Axioms (1)-(13) pertain to the fact that $(K, +, \cdot, ^\star, 0, 1)$ is a KA. While axioms (14)-(21) pertain to the fact that $(B, +, \cdot, ^-, 0, 1)$ is a Boolean algebra [24].

Example 1. *Semantically, programs are represented as binary relations over a set of states X and a test α is interpreted as a subset of the identity relation, comprising all pairs (x,x) such that α holds at state x. Hence,the family of binary relations on a set X is a KAT with operations defined as follows.*

$$0 \coloneqq \emptyset \qquad R + R' \coloneqq R \cup R'$$
$$1 \coloneqq \{(u,u) \mid u \in X\} \qquad R \cdot R' \coloneqq \{(u,w) \mid \exists v (u,v) \in R \wedge (v,w) \in R'\}$$
$$\overline{\alpha} = 1 \setminus \alpha \qquad R^\star \coloneqq \bigcup_{n \geq 0} R^n = \textit{reflexive transitive closure of } R$$

where $R^0 \coloneqq \{(u,u) \mid u \in X\}$ and $R^{n+1} = R \cdot R^n$.

This extension of KA with a Boolean algebra results in an algebraic model to capture program behavior and assertions. Hence, conditionals and **while** loops found in programming can be defined in terms of the regular operators.

$$\textbf{if } \alpha \textbf{ then } p \textbf{ else } q \stackrel{\text{def}}{=} \alpha \cdot p + \overline{\alpha} \cdot q$$
$$\textbf{while } \alpha \textbf{ do } p \stackrel{\text{def}}{=} (\alpha \cdot p)^\star \cdot \overline{\alpha}$$

2 Paraconsistent transition systems

The notion of *paraconsistent transition systems*, abbreviated to PLTS, was introduced in [17]. These systems' transitions involve two weights: a *positive* and a *negative* that characterize each transition in opposite ways. The positive weight represents the evidence of its presence, while the negative weight represents the evidence of its absence. Furthermore, following the line of research outlined in [8], a residuated lattice over a set A of possible truth values is adopted. This allows the transitions of PLTS to be represented by pairs of weights $(tt, ff) \in A \times A$. Thus, all the relevant constructions of PLTS are parametric in a class of residuated lattices and admit different instances according to the truth values domain A that better suits each concrete problem.

To exemplify, suppose that weights for both transitions come from a residuated lattice over the real interval $[0,1]$.

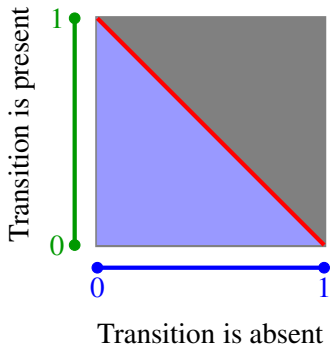


Figure 1: The vagueness-inconsistency square

These pairs of weights express different behaviors:

- *inconsistency*, when the positive and negative weights are contradictory,i.e., they sum to a value greater than 1, this corresponds to the upper triangle in Figure 1, filled in grey.
- *vagueness*, when the sum is less than 1, corresponding to the lower, periwinkle triangle in Figure 1.
- *consistency*, when the sum is exactly 1, that is the evidence degrees that enforce and prevent a transition from occurring are complementary, corresponding to the red line in Figure 1.

We will consider a class of residuated lattices $\boldsymbol{A} = \langle A, \sqcap, \sqcup, 1, 0, \rightarrow \rangle$ over a set A, bounded by a maximal element 1 and a minimal element 0 and where the lattice meet ($\sqcap$) and the monoidal composition ($\odot$) coincide. Such lattices are commonly referred to as *Heyting algebras* in the literature, and also known as pseudo-Boolean algebras [36]. The adjunction property is stated as $a \sqcap b \leq c$ if and only if $b \leq a \rightarrow c$. Finally, we will require that the Heyting algebras in the sequel be *complete*, i.e., every subset of A has both a greatest lower bound and a least upper bound.

Example 2. *The following lattices are complete Heyting algebras:*

- *the Boolean algebra* $\mathbf{2} = \langle \{0,1\}, \wedge, \vee, 1, 0, \rightarrow \rangle$
- *the Łukasiewicz three-valued algebra* $\mathbf{3} = \langle \{\top, u, \bot\}, \wedge_3, \vee_3, \top, \bot, \rightarrow_3 \rangle$*, where*

$\wedge_3$	$\bot$	u	$\top$
$\bot$	$\bot$	$\bot$	$\bot$
u	$\bot$	u	u
$\top$	$\bot$	u	$\top$

$\vee_3$	$\bot$	u	$\top$
$\bot$	$\bot$	u	$\top$
u	u	u	$\top$
$\top$	$\top$	$\top$	$\top$

$\rightarrow_3$	$\bot$	u	$\top$
$\bot$	$\top$	$\top$	$\top$
u	u	$\top$	$\top$
$\top$	$\bot$	u	$\top$

The truth value u stands for "unknown" and assumes different notations in the literature, such as $1/2$ *or #, which are interpreted as "possibility" or "indeterminacy" [35].*

- $\boldsymbol{G} = \langle [0,1], \min, \max, 0, 1, \rightarrow \rangle$*, with* $a \rightarrow b = 1$*, if* $a \leq b$ *and* $a \rightarrow b = b$ *otherwise.*

The following lemma combines [18, Lemma 1] and delineates several essential properties of complete residuated lattices, which we will resort to prove some results in this paper. For detailed proofs of Properties (22)-(24) and Properties (25)-(26), readers are referred to [18] and [8], respectively.

Lemma 1. *Let* $\boldsymbol{A}$ *be an complete Heyting algebra over a non empty set A. The following properties hold, for any* $a, a', b, b' \in A$

$$a \sqcap (b \sqcup b') = (a \sqcap b) \sqcup (a \sqcap b') \tag{22}$$

$$a \sqcup (b \sqcap b') = (a \sqcup b) \sqcap (a \sqcup b') \tag{23}$$

$$a \leq a' \text{ and } b \leq b' \text{ implies } a \sqcap b \leq a' \sqcap b' \tag{24}$$

$$a \sqcap \left(\bigsqcup_{i \in I} b_i \right) = \bigsqcup_{i \in I} (a \sqcap b_i) \tag{25}$$

$$\left(\bigsqcup_{i \in I} a_i \right) \sqcap b = \bigsqcup_{i \in I} (a_i \sqcap b) \tag{26}$$

where I is a (possibly infinite) index set.

Before providing a formal definition of PLTS, it is important to define the notion of a twisted structure. Initially introduced in Kracht's seminal work [31], a twisted structure results from the construction of the direct product of a residuated lattice $\boldsymbol{L}$ and its order-dual $\boldsymbol{L}^{\partial}$. This resulting lattice naturally possesses an involution[2] given by

$$/\!\!/(a,a') = (a',a)$$

for all $(a, a') \in \boldsymbol{L} \times \boldsymbol{L}^{\partial}$. Since its introduction, numerous authors have explored extensions of this structure by imposing additional properties on the residuated lattice [9, 10, 11]. Twisted structures play a fundamental role in the context of PLTS, as documented in prior work [16, 17, 19, 20], by enabling the computation of pairs of truth weights.

Definition 3. *Let $\boldsymbol{A} = \langle A, \sqcap, \sqcup, 1, 0, \rightarrow \rangle$ be a complete Heyting algebra. Its corresponding $\boldsymbol{A}$-twisted structure is denoted by $\mathcal{A} = \langle A \times A, \boxplus, \boxtimes, /\!\!/, (0,1), (1,0) \rangle$ and defined for any pair in $A \times A$ as:*

$$/\!\!/(a,b) = (b,a)$$
$$(a,a') \boxtimes (b,b') = (a \sqcap b, a' \sqcup b')$$
$$(a,a') \boxplus (b,b') = (a \sqcup b, a' \sqcap b')$$

The order in $\boldsymbol{A}$ is lifted to $\mathcal{A}$ as $(a,a') \preccurlyeq (b,b')$ iff $a \leq b$ and $a' \geq b'$.

As before, parentheses will be frequently omitted, reserving their use for enhancing the readability and clarity of certain expressions.

In the sequel, we will frequently refer to the next Lemma, which presents key properties of the operators defined above. Although most of these properties are evident, since the product of two complete, universally distributive lattices is itself a complete, universally distributive lattice, we explicitly state them for ease of reference in later sections. Lemma 2 essentially states that the twisted-structure in Definition 3 has the structure of a De Morgan algebra —i.e., a bounded distributive lattice in which $/\!\!/$ is involutive and satisfies De Morgan's laws (c.f. [25]). Additionally, it also states that the lattice $\langle A, \times A, \boxtimes, \boxplus \rangle$ is a quantale, that is, a complete lattice equipped with an associative operation $\boxtimes$ that satisfies distributive properties. This aligns with the inspiration for twisted structures drawn from Chu's work in category theory and its application to quantales [40].

Lemma 2. *Let $\mathcal{A}$ be a twisted structure, as defined in Definition 3. The following properties hold for any $(a,a'), (b,b'), (c,c') \in A \times A$.*

$$/\!\!/ /\!\!/ (a,a') = (a,a') \tag{27}$$

$$(a,a') \boxplus (1,0) = (1,0) \tag{28}$$

$$(a,a') \boxplus (0,1) = (a,a') \tag{29}$$

[2]Informally speaking, the operator $/\!\!/$ will denote a involutive negation in this paper and in essence it interchanges the positive and negative weight within a pair. See Example 3 for a concrete example.

$$(a,a') \boxplus (a,a') = (a,a') \tag{30}$$
$$(a,a') \boxcap (a,a') = (a,a') \tag{31}$$
$$(a,a') \boxplus (b,b') = (b,b') \boxplus (a,a') \tag{32}$$
$$(a,a') \boxcap (b,b') = (b,b') \boxcap (a,a') \tag{33}$$
$$/\!\!/((a,a') \boxplus (b,b')) = /\!\!/(a,a') \boxcap /\!\!/(b,b') \tag{34}$$
$$/\!\!/((a,a') \boxcap (b,b')) = /\!\!/(a,a') \boxplus /\!\!/(b,b') \tag{35}$$
$$(a,a') \boxcap (1,0) = (1,0) \boxcap (a,a') = (a,a') \tag{36}$$
$$(a,a') \boxcap (0,1) = (0,1) \boxcap (a,a') = (0,1) \tag{37}$$
$$(a,a') \boxplus \Big((b,b') \boxplus (c,c')\Big) = \Big((a,a') \boxplus (b,b')\Big) \boxplus (c,c') \tag{38}$$
$$(a,a') \boxcap \Big((b,b') \boxcap (c,c')\Big) = \Big((a,a') \boxcap (b,b')\Big) \boxcap (c,c') \tag{39}$$
$$(a,a') \boxcap \Big((b,b') \boxplus (c,c')\Big) = \Big((a,a') \boxcap (b,b')\Big) \boxplus \Big((a,a') \boxcap (c,c')\Big) \tag{40}$$
$$(a,a') \boxplus \Big((b,b') \boxcap (c,c')\Big) = \Big((a,a') \boxplus (b,b')\Big) \boxcap \Big((a,a') \boxplus (c,c')\Big) \tag{41}$$

Proof.

- Property (27) follows immediately since $/\!\!/$ is involutive.
- Property (28) and Property (29) are a consequence of 0 and 1 being the the least and greatest element of set A, respectively. Hence,

$$(a,a') \boxplus (1,0) = (a \sqcup 1, a' \sqcap 0) = (1,0)$$
$$(a,a') \boxplus (0,1) = (a \sqcup 0, a' \sqcap 1) = (a,a')$$

- Property (30) is a consequence of operators $\sqcap$ and $\sqcup$ being idempotent, $(a,a') \boxplus (a,a') = (a \sqcup a, a' \sqcap a') = (a,a')$. Property (31) follows similarly.
- Property (32) is a consequence of operators $\sqcap$ and $\sqcup$ being commutative. Consequently, it follows $(a,a') \boxplus (b,b') = (a \sqcup b, a' \sqcap b') = (b \sqcup a, b' \sqcap a') = (b,b') \boxplus (a,a')$. The proof for Property (33) follows similarly.
- The proof of Property (34) follows directly by the definition of the operators, $/\!\!/((a,a') \boxplus (b,b')) = (a' \sqcap b', a \sqcup b) = (a',a) \boxcap (b',b) = /\!\!/(a,a') \boxcap /\!\!/(b,b')$. Similarly, it is possible to prove Property (35).
- Property (36) since 0 and 1 are the least and greatest element of A, respectively, it follows $(1,0) \boxcap (a,a') = (1 \sqcap a, 0 \sqcup a') = (a,a')$ and by (33) it follows $(a,a') \boxcap (1,0) = (a,a')$. The proof of Property (37) follows similarly.
- To prove Property (38) note that,

$$(a,a') \boxplus ((b,b') \boxplus (c,c')) = (a,a') \boxplus (b \sqcup c, b' \sqcap c') = (a \sqcup (b \sqcup c), a' \sqcap (b' \sqcap c'))$$
$$((a,a') \boxplus (b,b')) \boxplus (c,c') = (a \sqcup b, a' \sqcap b') \boxplus (c,c') = ((a \sqcup b) \sqcup c, (a' \sqcap b') \sqcap c')$$

  Since $\sqcup$ and $\sqcap$ are associative it follows that $a \sqcup (b \sqcup c) = (a \sqcup b) \sqcup c$ and $a' \sqcap (b' \sqcap c') = (a' \sqcap b') \sqcap c'$. Therefore, operator $\boxplus$ is also associative. Similarly, it is possible to prove Property (39),i.e., $\boxcap$ is associative.

- Property (40) is a consequence of Property (22) and (23).

$$\begin{aligned}(a,a') \boxplus \Big((b,b') \boxplus (c,c') \Big) &= (a,a') \boxplus (b \sqcup c, b' \sqcap c') \\ &= (a \sqcap (b \sqcup c), a' \sqcup (b' \sqcap c')) \\ &= ((a \sqcap b) \sqcup (a \sqcap c), (a' \sqcup b') \sqcap (a' \sqcup c')) \\ &= (a \sqcap b, a' \sqcup b') \boxplus (a \sqcap c, a' \sqcup c') \\ &= \Big((a,a') \boxplus (b,b') \Big) \boxplus \Big((a,a') \boxplus (c,c') \Big)\end{aligned}$$

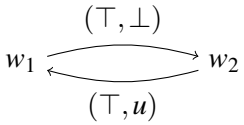
□

Finally, we present the definition of *paraconsistent transition systems* parametric in a class of residuated lattices over a set A of truth values.

Definition 4. *A paraconsistent transition system, abbreviated to PLTS, is a tuple $M = (W,R,V)$ such that:*

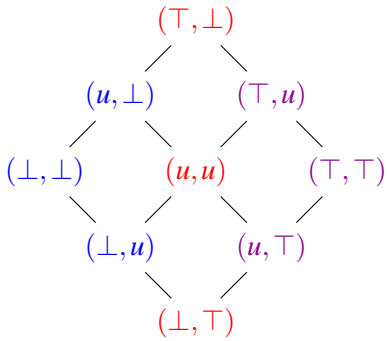
- *W is a non empty set of states,*
- *$R\colon W \times W \to A \times A$ is a paraconsistent accessibility relation. For any pair of states $(w_1,w_2) \in W \times W$ relation R assigns a pair $(tt,ff) \in A \times A$ where tt represents the evidence of the transition from w_1 to w_2 occurring and ff represents the evidence of being prevented from occurring.*
- *$V : W \times Prop \to A \times A$ is a valuation function, that assigns to a proposition symbol p at a given state w a pair $(tt,ff) \in A \times A$ such that tt is the evidence of p holding at w and ff the evidence of not holding*

Example 3. *Consider the residuated lattice* **3** *and the set of proposition symbols $\{p\}$. The following model $M = (\{w_1,w_2\},R,V)$ is a PLTS where*



$$\begin{aligned}V :\ & W \times \{p\} \to \{\top,u,\bot\} \times \{\top,u,\bot\} \\ & (w_1,p) \mapsto (\top,\bot) \\ & (w_2,p) \mapsto (u,\bot)\end{aligned}$$

We consider, following Łukasiewicz , that the truth value u is an intermediate value between $\top$ and $\bot$ [35]. Hence, a natural "ordering" of the three values is $\bot \leq u \leq \top$. In this example, each weight takes values in the set $\{\bot,u,\top\}$, making it possible to represent the resulting lattice of doing the direct product of the three-valued lattice and its order-dual.



All pairs marked in red represent consistent information. The pairs on the left, shown in blue, represent vague information, while those on the right, highlighted in magenta, represent inconsistent information.

Let us observe the intuition behind some of the operators in the twisted structure defined in Definition 3. For example, $V(w_2, p) = (u, \bot)$ indicates that at state w_2, there is uncertain evidence (u) that p holds and minimal evidence ($\bot$) that p does not hold. Similarly, one could say that at state w_2, there is minimal evidence that the negation of p holds and uncertain evidence that it does not hold. This reflects the intuition behind the operator $/\!\!/$, which acts as an involutive negation by switching the positive and negative weights.

Considering the valuations of p at states w_1 and w_2, it is possible to compute the evidence of p holding or not at both states. The certainty that p holds in both states is equal to the certainty that it holds in each state ($\top \sqcap u$). Similarly, the certainty that p does not hold in both states is equal to the certainty that it does not hold in either state ($\bot \sqcup \bot$). This captures the intuition behind the operator $\sqcap\!\!\sqcap$.

The interested reader is referred to [20] for a more detailed exploration of the paraconsistent logic underlying PLTS.

3 Paraconsistent Kleene Algebra with tests

The approach presented in this work aims to reason about program executions in a paraconsistent manner, where executions and tests may involve vagueness as well as inconsistencies. Consequently, rather than yielding a bivalent outcome as in traditional KAT, the outcome is graded by a pair of weights, one weight indicates the evidence for execution and the other the evidence for failure. Such framework entails the need to weaken the Boolean subalgebra of KAT which leads to the following variant:

Definition 5. *A* paraconsistent Kleene algebra with tests *(PKAT) is a tuple*

$$(K, T, +, \cdot, ^\star, ^-, 0, 1)$$

such that $(K, +, \cdot, ^\star, 0, 1)$ is a Kleene algebra, $(T, +, \cdot, ^-, 0, 1)$ satisfies axioms (14)-(19) *and $B \subseteq T$. Relation $\leq$ is induced by $+$, that is, $p \leq q$ iff $p + q = q$.*

A key aspect of the KAT axiomatization lies in axioms (20) and (21), which informally express the principles of non-contradiction and excluded middle, respectively. These two principles play a significant role in the philosophy of paraconsistency, which rejects the principle of non-contradiction, and in intuitionism, which does not assume the principle of the excluded middle. The notion of PKAT takes in consideration these philosophies and consequently forms a weakened version of KAT by only rejecting axioms (20) and (21). Hence,

Theorem 1. *Any KAT is a PKAT.*

Proof. By definition, any KAT satisfies axioms (1)-(19). Thus, trivially any KAT is a PKAT. □

The weakening discussed in this paper generalizes Boolean algebras, in that Boolean algebras are precisely those algebras that satisfy both the principle of non-contradiction (20) and the principle of the excluded middle (21). This generalization is similar to that presented by Heyting algebras [7], where any Boolean algebra is a Heyting algebra that satisfies the principle of the excluded middle. In fact, the weakening presented in this paper is a stronger generalization than Heyting algebras, as any algebra failing to satisfy both principles inherently does not satisfy the principle of the excluded middle. A potential implication of this observation is that the discussed weakening of the Boolean algebra could

potentially serve as algebraic models for propositional paraconsistent logic, much like how Heyting algebras model propositional intuitionistic logic [6] and Boolean algebras model propositional classical logic.

In the remaining of this section we introduce two examples of algebras where program executions and tests may encompass inconsistencies and vagueness. Consequently, these algebras can be formalized as PKAT. To achieve this, we refer back to Definition 3, which establishes that for any complete Heyting algebra $\boldsymbol{K}$ over a non-empty set K of possible truth values, its corresponding twisted structure $\mathcal{K}$ allows for the computation of pairs of truth values $K \times K$.

Remark. *Given sets T and W, we denote by $(T \times T)^W$ the set of functions $W \to (T \times T)$.*

Definition 6. *Let W be a set and $\boldsymbol{T}$ be a complete Heyting algebra over a non empty set of truth values T. The algebra of paraconsistent sets over the twisted-structure $\mathcal{T}$ is*

$$\mathbf{Set_P}(\mathcal{T}) = \langle (T \times T)^W, (T \times T)^W, +, \cdot, ^\star, ^-, \oslash, \Upsilon \rangle$$

For any paraconsistent sets over W, $\varphi, \psi \in (T \times T)^W$ and $w \in W$, operators are defined pointwise by

$$\begin{aligned} \oslash(w) &= (0,1) & (\varphi + \psi)(w) &= \varphi(w) \sqcup \psi(w) \\ \Upsilon(w) &= (1,0) & (\varphi \cdot \psi)(w) &= \varphi(w) \sqcap \psi(w) \\ \overline{\varphi}(w) &= /\!\!/\varphi(w) & (\varphi^\star)(w) &= \bigsqcup_{n \geq 0} \varphi^n(w) \end{aligned}$$

with $\varphi^0(w) = \Upsilon(w)$ and $\varphi^{n+1}(w) = (\varphi \cdot \varphi^n)(w)$. The values of paraconsistent sets $\varphi(w)$ and $\psi(w)$ are elements of $T \times T$, and constants $\oslash$ and Υ are the least and the greatest elements of $T \times T$, respectively. The partial order in paraconsistent sets φ and ψ in $(T \times T)^W$ is given by

$$\varphi \subseteq \psi \text{ if and only if } \forall w \in W,\ \varphi(w) \preccurlyeq \psi(w)$$

Example 4. *Consider the set $W = \{w_1, w_2\}$ and the three-valued residuated lattice $\mathbf{3} = \{\top, u, \bot\}$. Let $\varphi, \psi \in \mathbf{Set_P}(\mathbf{3})$ be two paraconsistent sets defined as*

$$\begin{aligned} \varphi\colon\ & W \to \mathbf{3} \times \mathbf{3} & \psi\colon\ & W \to \mathbf{3} \times \mathbf{3} \\ & w_1 \mapsto (\top, u) & & w_1 \mapsto (\top, \bot) \\ & w_2 \mapsto (u, u) & & w_2 \mapsto (\top, u) \end{aligned}$$

It is possible to define with operator $^-$ two other paraconsistent sets denoted by $\overline{\varphi}, \overline{\psi} \in \mathbf{Set_P}(\mathbf{3})$ defined as,

$$\begin{aligned} \overline{\varphi}\colon\ & W \to \mathbf{3} \times \mathbf{3} & \overline{\psi}\colon\ & W \to \mathbf{3} \times \mathbf{3} \\ & w_1 \mapsto (u, \top) & & w_1 \mapsto (\bot, \top) \\ & w_2 \mapsto (u, u) & & w_2 \mapsto (u, \top) \end{aligned}$$

Note that $\varphi \subseteq \psi$, while $\overline{\psi} \subseteq \overline{\varphi}$.

Theorem 2. *For any complete Heyting algebra $\boldsymbol{T}$ over a non empty set T of possible truth values, $\mathbf{Set_P}(\mathcal{T})$ forms a PKAT.*

Proof. For a fixed complete Heyting algebra $\boldsymbol{T}$, by Definition 3, we define its twisted structure $\mathcal{T}$. We will prove that $\mathbf{Set_P}(\mathcal{T})$ defined as in Definition 6 forms a PKAT, that is, we show that axioms (1)-(19) are satisfied.

Axiom (1) by Property (38), $\varphi(w) \sqcup (\psi(w) \sqcup \phi(w)) = (\varphi(w) \sqcup \psi(w)) \sqcup \phi(w)$. Using the definition of

$+$, it follows that $(\varphi+(\psi+\phi))(w) = ((\varphi+\psi)+\phi)(w)$.
Axiom (2) by Property (32) and the definition of $+$, $(\varphi+\psi)(w) = (\psi+\varphi)(w)$.
Axiom (3) from Property (29) it follows that, $(\varphi+\oslash)(w) = \varphi(w) \sqcup \oslash(w) = \varphi(w) \sqcup (0,1) = \varphi(w)$.
Axiom (4) is immediate by Property (30), $(\varphi+\varphi)(w) = \varphi(w)$
Axiom (5) is a consequence of Property (39), $\varphi(w) \sqcap (\psi(w) \sqcap \phi(w)) = (\varphi(w) \sqcap \psi(w)) \sqcap \phi(w)$. Using the definition of $\cdot$, $(\varphi \cdot (\psi \cdot \phi))(w) = ((\varphi \cdot \psi) \cdot \phi)(w)$.
Axiom (6) from Property (36) it follows that $(\Upsilon \cdot \varphi)(w) = (\varphi \cdot \Upsilon)(w) = \varphi(w)$.
Axiom (7) using Property (40) and the definition of $+$ and $\cdot$ it follows that, $(\varphi \cdot (\psi+\phi))(w) = ((\varphi \cdot \psi)+(\varphi \cdot \phi))(w)$. Axiom (8) follows by Property (33).
Axiom (9) by Property (37) it follows that $(\oslash \cdot \varphi)(w) = (\varphi \cdot \oslash)(w) = \oslash(w)$.
Axiom (10) can be derived as follows,

$$\begin{aligned}
(\Upsilon + (\varphi \cdot \varphi^\star))(w) =& \Upsilon(w) \sqcup \left(\varphi(w) \sqcap \bigsqcup_{n\geq 0} \varphi^n \right) \\
=& \varphi^0(w) \sqcup \left(\varphi(w) \sqcap (\varphi^0(w) \sqcup \varphi(w) \sqcup \varphi^2(w) \ldots) \right) \\
& \{\text{using (40) and defn. of } ^\star\} \\
=& \varphi^0(w) \sqcup (\varphi(w) \sqcup \varphi^2(w) \sqcup \ldots) \\
=& \varphi^\star(w)
\end{aligned}$$

Similarly, it is possible to show Axiom (11).
Axiom (12) Let us start by assuming that $(\varphi \cdot \psi)(w) \preccurlyeq \psi(w)$. Then,

$$\begin{aligned}
(\varphi^\star \cdot \psi)(w) =& \left(\varphi^0(w) \sqcup \varphi(w) \sqcup \varphi^2(w) \sqcup \ldots \right) \sqcap \psi(w) \\
& \{\text{using (33), (40) and (36)}\} \\
=& \psi(w) \sqcup (\varphi(w) \sqcap \psi(w)) \sqcup (\varphi^2(w) \sqcap \psi(w)) \sqcup \ldots \\
=& \psi(w) \sqcup (\varphi \cdot \psi)(w) \sqcup (\varphi^2 \cdot \psi)(w) \sqcup \ldots \qquad (42)
\end{aligned}$$

For any integer $n \geq 0$, $(\varphi^n \cdot \psi)(w) = \underbrace{\varphi \cdot \ldots \cdot \varphi}_{n} \cdot \psi(w)$. Using the hypothesis n times, it follows that $(\varphi^n \cdot \psi)(w) \preccurlyeq \psi(w)$. Hence, by (42) and given that $(\psi \cdot \psi)(w) \preccurlyeq \psi(w)$ it follows that

$$\psi(w) \sqcup (\varphi \cdot \psi)(w) \sqcup (\varphi^2 \cdot \psi)(w) \sqcup \ldots \preccurlyeq \psi(w)$$

Similarly, it is possible to show Axiom (13). Axiom (14) $(\varphi+(\psi \cdot \phi))(w) = ((\varphi+\psi) \cdot (\varphi+\phi))(w)$ follows by Property (41) and the definition of $\cdot$ and $+$. Similarly, Axiom (15) follows from Property (30) and (41). Axiom (16) results from Property (33), $(\varphi \cdot \psi)(w) = (\psi \cdot \varphi)(w)$. Finally, it is possible to show Axiom (17), that is, $(\varphi \cdot \varphi)(w) = \varphi(w)$; Axiom (18), that is, $\overline{\overline{\varphi}}(w) = \varphi(w)$ and Axiom (19), that is, $(\varphi+\Upsilon)(w) = \Upsilon(w)$ follow directly from Property (31), (27) and (28), respectively. □

The aim of the following definition is to explore paraconsistent programs similar to the well-known binary programs presented in Example 1. These paraconsistent programs involve computations that may exhibit vagueness or inconsistency. Thus potentially lending themselves to representation as PLTS with transitions and valuations weighted by pairs of weights, tailored to fit the specific problem domain. Paraconsistent relations are defined over a pair of states $W \times W$, where a test α can be interpreted at any

state $w \in W$. Specifically, if a test α is evaluated by a pair (tt, ff) at state w , then tt the evidence of the test holding and ff measures the evidence of the test not holding at state w. Let's proceed to define the algebra of paraconsistent relations.

Definition 7. *Let W be a set, $\boldsymbol{K}$ and $\boldsymbol{T}$ be complete Heyting algebras over a non empty set of truth values K and T, respectively. The algebra of paraconsistent relations over $\mathcal{K}$ and $\mathcal{T}$ is defined as*

$$\mathbf{Rel_P}(\mathcal{K},\mathcal{T}) = \langle (K\times K)^{W\times W}, (T\times T)^{W\times W}, +, \cdot, ^\star, ^-, \oslash, \Lambda\rangle$$

where $(K\times K)^{W\times W}$ is the set of all paraconsistent relations over $W\times W$, i.e. functions $(W\times W)\to(K\times K)$. The elements of $(T\times T)^{W\times W}$ are paraconsistent tests *t such that $t(u,v) = (0,1)$ whenever $u \neq v$. The operators of paraconsistent relations are defined pointwise by*

$$\oslash(u,v) = (0,1) \qquad (R+R')(u,v) = R(u,v) \amalg R'(u,v)$$

$$\Lambda(u,v) = \begin{cases} (1,0) & \text{if } u = v \\ (0,1) & \text{otherwise} \end{cases} \qquad (R\cdot R')(u,v) = \coprod_{w\in W} (R(u,w) \sqcap R'(w,v))$$

$$\overline{t}(u,v) = /\!\!/ t(u,v) \qquad R^\star(u,v) = \coprod_{n\geq 0} R^n(u,v)$$

with $R^{n+1}(u,v) = (R\cdot R^n)(u,v)$ and $R^0(u,v) = \Lambda(u,v)$. The value of paraconsistent relations, $R(u,v)$ and $R'(u,v)$ are elements of $K\times K$, the value of $t(u,v)$ is an element of $T\times T$, and constants $\oslash, \Lambda$ are the least and the greatest elements of $T\times T$. The partial order $\subseteq$ for paraconsistent relations is given by

$$R \subseteq R' \text{ if and only if } \forall u, v \in W, R(u,v) \preccurlyeq R'(u,v)$$

Theorem 3. *Let $\boldsymbol{K}$ and $\boldsymbol{T}$ be complete Heyting algebras over a non empty set K and T of possible truth values such that $T \subseteq K$, $\mathbf{Rel_P}(\mathcal{K},\mathcal{T})$ forms a* PKAT.

Proof. Let $\boldsymbol{K}$ and $\boldsymbol{T}$ be complete Heyting algebras over set K and T, respectively, such that $T \subseteq K$. It is possible to define the corresponding twisted structures $\mathcal{K}$ and $\mathcal{T}$ as described in Definition 3. We will prove that $\mathbf{Rel_P}(\mathcal{K},\mathcal{T})$ forms a PKAT, that is, axioms (1)-(19) are satisfied. The satisfaction of axioms (1)-(4) and (18) is similar to Theorem 2. Let us show the remaining.
Axiom (5) derives as follows

$$\begin{aligned}
(R\cdot(R'\cdot R''))(w,v) &= \coprod_{u\in W}\left(R(w,u)\sqcap \coprod_{t\in W}\left(R'(u,t)\sqcap R''(t,v)\right)\right)\\
&= \coprod_{u\in W}\coprod_{t\in W}\left(R(w,u)\sqcap R'(u,t)\sqcap R''(t,v)\right)\\
&= \coprod_{t\in W}\left(\coprod_{u\in W}\left(R(w,u)\sqcap R'(u,t)\right)\sqcap R''(t,v)\right)\\
&= \coprod_{u\in W}\left((R\cdot R')(w,t)\sqcap R''(t,v)\right)\\
&= ((R\cdot R')\cdot R'')(w,v)
\end{aligned}$$

Axiom (6) by definition of $\cdot$, $(\Lambda\cdot R)(w,v) = \coprod_{u\in W}\left(\Lambda(w,u)\sqcap R(u,v)\right)$.
For all $u \neq w$, $\Lambda(w,u) = (0,1)$ and by (37) it follows, $\Lambda(w,u)\sqcap R(u,v) = (0,1)$. Futhermore, by

Property (28) it follows that $(\Lambda \cdot R)(w,v) = \Lambda(w,w) \sqcap R(w,v) = (1,0) \sqcap R(w,v) = R(w,v)$.
Axiom (7) proceeds as

$$\begin{aligned}(R \cdot (R' + R''))(w,v) =& \bigsqcup_{u \in W} (R(w,u) \sqcap (R'(u,v) \sqcup R''(u,v)))\\ &\{\text{using Property (40)}\}\\ =& \bigsqcup_{u \in W} \left(\left(R(w,u) \sqcap R'(u,v) \right) \sqcup \left(R(w,u) \sqcap R''(u,v) \right) \right)\\ =& \bigsqcup_{u \in W} \left(R(w,u) \sqcap R'(u,v) \right) \sqcup \bigsqcup_{u \in W} \left(R(w,u) \sqcap R''(u,v) \right)\\ =& (R \cdot R')(w,v) + (R \cdot R'')(w,v)\end{aligned}$$

Axiom (8) follows similarly by Property (33).
Axiom (9) by Property (37) it follows that,

$$(\oslash \cdot R)(w,v) = \bigsqcup_{u \in W} \left(\oslash(w,u) \sqcap R(u,v) \right) = \bigsqcup_{u \in W} \left((0,1) \sqcap R(u,v) \right) = (0,1) = \oslash(w,v)$$

Consequently, since $\sqcap$ is commutative, it follows $(R \cdot \oslash)(w,v) = \oslash(w,v)$.
Similar to Axiom (11), Axiom (10) derives from,

$$\begin{aligned}(\Lambda + (R \cdot R^*))(w,v) =& \Lambda(w,v) \sqcup \left(\bigsqcup_{u \in W} R(w,u) \sqcap \left(\bigsqcup_{n \geq 0} R^n(u,v) \right) \right)\\ &\{\text{using Property (40)}\}\\ =& \Lambda(w,v) \sqcup \left(\bigsqcup_{n \geq 0} \left(\bigsqcup_{u \in W} R(w,u) \sqcap R^n(u,v) \right) \right)\\ =& R^0(w,v) \sqcup \left(\bigsqcup_{n \geq 0} (R \cdot R^n)(w,v) \right)\\ =& R^0(w,v) \sqcup \left(\bigsqcup_{n > 0} R^n(w,v) \right)\\ =& R^*(w,v)\end{aligned}$$

Axiom (12) Let us assume that $(R \cdot R')(w,v) \preccurlyeq R'(w,v)$. Then,

$$\begin{aligned}(R^* \cdot R')(w,v) =& \bigsqcup_{u \in W} \left(\bigsqcup_{n \geq 0} R^n(w,u) \sqcap R'(u,v) \right)\\ =& \bigsqcup_{n \geq 0} \left(\bigsqcup_{u \in W} (R^n(w,u) \sqcap R'(u,v)) \right)\\ =& \bigsqcup_{n \geq 0} (R^n \cdot R')(w,v)\\ =& R'(w,v) \sqcup (R \cdot R')(w,v) \sqcup (R^2 \cdot R')(w,v) \sqcup \ldots\\ &\{\text{Hypothesis}\}\\ \preccurlyeq& R'(w,v)\end{aligned}$$

Similarly, it is possible to show Axiom (13).
Axiom (14) derives as,

$$\begin{aligned}(t+(t'\cdot t''))(u,v) =& t(u,v) \mathbin{\mathbb{U}} \left(\bigsqcup_{w\in W} (t'(u,w) \mathbin{\sqcap} t''(w,v)) \right) \\ & \{(\text{step } \star)\} \\ =& t(u,v) \mathbin{\mathbb{U}} \left(t'(u,v) \mathbin{\sqcap} t''(u,v) \right) \\ & \{\text{using Property (41)}\} \\ =& \left(t(u,v) \mathbin{\mathbb{U}} t'(u,v) \right) \mathbin{\sqcap} \left(t(u,v) \mathbin{\mathbb{U}} t''(u,v) \right) \\ =& (t+t')(u,v) \mathbin{\sqcap} (t+t'')(u,v)\end{aligned}$$

(step $\star$) for any $w \in W$, $(t'(u,w) \mathbin{\sqcap} t''(w,v)) \neq (0,1)$, iff $(t'(u,w) \neq (0,1)$ and $t''(w,v) \neq (0,1))$. Thus, $(t'(u,w) \mathbin{\sqcap} t''(w,v)) \neq (0,1)$ only when $w = u$ and $w = v$.
Also note that,

$$\begin{aligned}((t+t')\cdot(t+t''))(u,v) =& \bigsqcup_{w\in W} \left(\left(t(u,w) \mathbin{\mathbb{U}} t'(u,w) \right) \mathbin{\sqcap} \left(t(w,v) \mathbin{\mathbb{U}} t''(w,v) \right) \right) \\ & \{(\text{step } \star\star)\} \\ =& \left(t(u,v) \mathbin{\mathbb{U}} t'(u,v) \right) \mathbin{\sqcap} \left(t(u,v) \mathbin{\mathbb{U}} t''(u,v) \right) \\ =& (t+t')(u,v) \mathbin{\sqcap} (t+t'')(u,v)\end{aligned}$$

(step $\star\star$) for any $w \in W$,

$$\left(t(u,w) \mathbin{\mathbb{U}} t'(u,w) \right) \mathbin{\sqcap} \left(t(w,v) \mathbin{\mathbb{U}} t''(w,v) \right) \neq (0,1)$$

if and only if $(t(u,w) \mathbin{\mathbb{U}} t'(u,w) \neq (0,1)$ and $t(w,v) \mathbin{\mathbb{U}} t''(w,v) \neq (0,1))$ Hence, $t(u,w)$, $t'(u,w)$, $t(w,v)$ and $t''(w,v)$ must all be different from $(0,1)$ which implies $u = w = v$.
Therefore, we show that $(t+(t'\cdot t''))(u,v) = ((t+t')\cdot(t+t''))(u,v)$. By Property (30) it is possible to show Axiom (15).
Axiom (16) by definition,

$$\begin{aligned}(t\cdot t')(u,v) &= \bigsqcup_{w\in W} \left(t(u,w) \mathbin{\sqcap} t'(w,v) \right) \\ (t'\cdot t)(u,v) &= \bigsqcup_{w\in W} \left(t'(u,w) \mathbin{\sqcap} t(w,v) \right)\end{aligned}$$

Whenever $w \neq u$ or $w \neq v$, by definition of test and by (37), it follows that $t(u,w) \mathbin{\sqcap} t'(w,v) = t'(u,w) \mathbin{\sqcap} t(w,v) = (0,1)$. Hence, $t(u,w) \mathbin{\sqcap} t'(w,v) \neq (0,1)$ and $t'(u,w) \mathbin{\sqcap} t(w,v) \neq (0,1)$ only when $w = u$ and $w = v$. Thus,

$$(t\cdot t')(u,v), = t(u,v) \mathbin{\sqcap} t'(u,v) = t'(u,v) \mathbin{\sqcap} t(u,v) = (t'\cdot t)(u,v)$$

Axiom (17) by definition of $\cdot$, $(t\cdot t)(u,v) = \bigsqcup_{w\in W} \left(t(u,w) \mathbin{\sqcap} t(w,v) \right)$.
Since $t(u,w) \mathbin{\sqcap} t(w,v) = (0,1)$ whenever $u \neq w$ or $w \neq v$. Hence, by (29) and (31) $(t\cdot t)(u,v) = t(u,v) \mathbin{\sqcap}$

$t(u,v) = t(u,v)$.
Axiom (19) By definition of $+$, $(t+\Lambda)(u,v) = t(u,v) + \Lambda(u,v)$. If $u = v$ then, $\Lambda(u,v) = (1,0)$ and by Property (28) $(t+\Lambda)(u,v) = (1,0) = \Lambda(u,v)$. Otherwise, $t(u,v) = (0,1)$ and by Property (29), $(t+\Lambda)(u,v) = \Lambda(u,v)$. Hence, $(t+\Lambda)(u,v) = \Lambda(u,v)$. □

4 Conclusion and future work

This paper contributes to an ongoing research agenda focused on paraconsistent transition systems (PLTS) and their logics. PLTS were initially introduced in [17], followed by the introduction of a logic to express their properties in [16] and the respective application to the analysis of quantum circuits was further discussed in [5]. Subsequently, the algebra of constructors and abstractors for PLTS was discussed in [19], which served as the foundation for a structured specification theory outlined in [20].

Here our focus is on developing an algebraic counterpart to this line of work. Hence, we take the initial steps towards introducing a variant of KAT to reason about vague and inconsistent computations and assertions. A similar roadmap for reasoning about fuzzy computations can be found in [23]. As in [23], given that such assertions often take the form of tests, our approach lies in the modification of KAT that deals with properties of tests. The approach taken in this paper rejects the principle of non-contradiction and the principle of the excluded middle; consequently, some classical properties of Boolean algebra are lost. The resulting PKAT can interpret computations entailing contradictions or vagueness.

A possible application of this work is in quantum circuits where a phenomenon known as *decoherence* can occur. Such phenomenon is characterized by the loss of information from the circuit due to unwanted interaction with the environment. When the coherence time of a qubit is exceeded, there is an increasing probability that the circuit does not behave according to its design. Typically, qubit coherence is not specified exactly but is given as time intervals in the literature, corresponding to worst-case and best-case scenarios. In [17], it is proposed to use the two accessibility relations in PLTS to model these scenarios simultaneously. The minimum coherence time $t_{\min}$ determines the negative weight, interpreted as the likelihood that the system evolves to a decoherent state, and the maximum coherence time $t_{\max}$ determines the positive weight, interpreted as the likelihood that the system remains coherent. With further considerations, it becomes feasible to translate quantum circuits into PLTS, as elaborated in [5, 17].

The parametric approach adopted in this work is based on prior work documented in [8], which has been applied to formalize paraconsistent transition systems and their corresponding logics in [16, 19]. Given two residuated lattices $\boldsymbol{K}$ and $\boldsymbol{T}$ over a set of possible truth values K and T such that $T \subseteq K$, it is possible to define a twisted structure to operate on pairs $K \times K$. This structure allows for the introduction of two algebras in this paper: $\mathbf{Set_P}(\mathcal{T})$ and $\mathbf{Rel_P}(\mathcal{K},\mathcal{T})$ parametric to the twisted structures. The main results demonstrate that both algebras $\mathbf{Set_P}(\mathcal{T})$ and $\mathbf{Rel_P}(\mathcal{K},\mathcal{T})$ form a PKAT, Theorem 2 and 3.

Since KAT provides a framework for reasoning about imperative programs in a (quasi) equational way, we aim to explore an encoding of propositional Hoare logic into PKAT. However, for such task it may be necessary to refine PKAT with additional properties and both the meaning of Hoare triples and the inference rules need adjustment. Similarly to [23], we propose encoding a Hoare triple $\{b\}p\{c\}$ in PKAT as $b \cdot p \preccurlyeq b \cdot p \cdot c$, conveying that program correctness can only improve with execution.

The study of the languages underlying paraconsistent transition structures is a line of work to be explored soon. In analogy to what is done in classic automata theory, we will consider a notion of "paraconsistent automata", by enriching the PLTS structure with a set of accepting states, in order to generate and characterize the algebra of the recognized expressions, say the paraconsistent regular languages. At this level, we expect to establish a Kleene-like Theorem and to frame such languages algebras as a PKAT.

Additionally, it is necessary to conduct a more thorough investigation into the potential applications and limitations that may arise from the flexibility of the adopted approach.

References

[1] Jair Minoro Abe, Cláudio Rodrigo Torres, Germano Lambert-Torres, João Inácio da Silva Filho & Helga Gonzaga Martins (2007): *Paraconsistent Autonomous Mobile Robot Emmy III*. In Germano Lambert-Torres, Jair Minoro Abe, João Inácio da Silva Filho & Helga Gonzaga Martins, editors: *Advances in Technological Applications of Logical and Intelligent Systems, Selected Papers from the Sixth Congress on Logic Applied to Technology, LAPTEC 2007, Unisanta, Santa Cecilia University, Santos, Brazil, November 21-23, 2007*, *Frontiers in Artificial Intelligence and Applications* 186, IOS Press, pp. 236–258, doi:10.3233/978-1-58603-936-3-236.

[2] Juan Carlos Agudelo & Walter Alexandre Carnielli (2010): *Paraconsistent Machines and their Relation to Quantum Computing*. *J. Log. Comput.* 20(2), pp. 573–595. Available at https://doi.org/10.48550/arXiv.0802.0150.

[3] Seiki Akama, editor (2016): *Towards Paraconsistent Engineering*. *Intelligent Systems Reference Library* 110, Springer, doi:10.1007/978-3-319-40418-9.

[4] Roland C. Backhouse, Dexter Kozen & Bernhard Möller (2002): *Applications of Kleene Algebra (Dagstuhl Seminar 01081)*. Dagstuhl Seminar Report 298, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, doi:10.4230/DagSemRep.298.

[5] Luís Soares Barbosa & Alexandre Madeira (2023): *Capturing Qubit Decoherence through Paraconsistent Transition Systems*. In Shigeru Chiba, Youyou Cong & Elisa Gonzalez Boix, editors: *Companion Proceedings of the 7th International Conference on the Art, Science, and Engineering of Programming, Programming 2023, Tokyo, Japan, March 13-17, 2023*, ACM, pp. 109–110, doi:10.1145/3594671.3594689.

[6] Nick Bezhanishvili & Dick de Jongh (2005): *Intuitionistic Logic*. *ESSLLI course notes*. Available at https://www.math.uni-hamburg.de/en/personen/khomskii/intuitionistic/PP-2006-25.text.pdf.

[7] Francis Borceux (1994): *Locales*, p. 1–86. Encyclopedia of Mathematics and its Applications, Cambridge University Press. Available at https://www.cambridge.org/core/books/handbook-of-categorical-algebra/A0B8285BBA900AFE85EED8C971E0DE14.

[8] Félix Bou, Francesc Esteva, Lluís Godo & Ricardo Oscar Rodríguez (2009): *On the Minimum Many-Valued Modal Logic over a Finite Residuated Lattice*. *Journal of Logic and Computation* 21(5), pp. 739–790, doi:10.1093/logcom/exp062.

[9] Manuela Busaniche & Roberto Cignoli (2014): *The subvariety of commutative residuated lattices represented by twist-products*. *algebra universalis* 71, doi:10.1007/s00012-014-0265-4.

[10] Manuela Busaniche, Nikolaos Galatos & Miguel Andrés Marcos (2022): *Twist Structures and Nelson Conuclei*. *Stud Logica* 110(4), pp. 949–987, doi:10.1007/S11225-022-09988-Z.

[11] Jose Castiglioni, M. Menni & Marta Sagastume (2008): *On Some Categories of Involutive Centered Residuated Lattices*. *Studia Logica* 90, pp. 93–124, doi:10.1007/s11225-008-9145-2.

[12] Maria Luisa Dalla Chiara & Roberto Giuntini (2000): *Paraconsistent ideas in quantum logic*. *Synth.* 125(1-2), pp. 55–68. Available at https://doi.org/10.1023/A:1005296018904.

[13] J.H. Conway (1971): *Regular Algebra and Finite Machines*. Chapman and Hall mathematics series, Chapman and Hall. Available at https://books.google.pt/books?id=xBXvAAAAMAAJ.

[14] Newton C. A. Costa & Walter A. Carnielli (1986): *On Paraconsistent Deontic Logic*. *Philosophia* 16(3-4), pp. 293–305, doi:10.1007/bf02379748.

[15] Newton C. A. Da Costa & E. H. Alves (1977): *A Semantical Analysis of the Calculi C N*. *Notre Dame Journal of Formal Logic* 18(4), pp. 621–630, doi:10.1305/ndjfl/1093888132.

[16] Ana Cruz, Alexandre Madeira & Luís Soares Barbosa (2022): *A Logic for Paraconsistent Transition Systems*. In Andrzej Indrzejczak & Michal Zawidzki, editors: *10th International Conference on Non-Classical Logics. Theory and Applications*, *EPTCS* 358, pp. 270–284. Available at https://doi.org/10.4204/EPTCS.358.20.

[17] Ana Cruz, Alexandre Madeira & Luís Soares Barbosa (2022): *Paraconsistent Transition Systems*. In Daniele Nantes-Sobrinho & Pascal Fontaine, editors: *Proceedings 17th International Workshop on Logical and Semantic Frameworks with Applications, LSFA 2022, Belo Horizonte, Brazil (hybrid), 23-24 September 2022*, *EPTCS* 376, pp. 3–15. Available at https://doi.org/10.4204/EPTCS.376.3.

[18] Juliana Cunha, Alexandre Madeira & Luis S. Barbosa (Available here): *Paraconsistent transition structures: compositional principles and a modal logic*. *(submitted to a journal)*.

[19] Juliana Cunha, Alexandre Madeira & Luís Soares Barbosa (2023): *Stepwise Development of Paraconsistent Processes*. In Cristina David & Meng Sun, editors: *Theoretical Aspects of Software Engineering - 17th International Symposium, TASE 2023, Bristol, UK, July 4-6, 2023, Proceedings*, *Lecture Notes in Computer Science* 13931, Springer, pp. 327–343. Available at https://doi.org/10.1007/978-3-031-35257-7_20.

[20] Juliana Cunha, Alexandre Madeira & Luís Soares Barbosa (2023): *Structured Specification of Paraconsistent Transition Systems*. In Hossein Hojjat & Erika Ábrahám, editors: *Fundamentals of Software Engineering - 10th International Conference, FSEN 2023, Tehran, Iran, May 4-5, 2023, Revised Selected Papers*, *Lecture Notes in Computer Science* 14155, Springer, pp. 1–17. Available at https://doi.org/10.1007/978-3-031-42441-0_1.

[21] Hyghor Miranda Côrtes, Paulo Eduardo Santos & João Inácio da Silva Filho (2022): *Monitoring electrical systems data-network equipment by means of Fuzzy and Paraconsistent Annotated Logic*. *Expert Systems with Applications* 187, p. 115865. Available at https://doi.org/10.1016/j.eswa.2021.115865.

[22] Michael Dunn & Greg Restall (2002): *Relevance Logic*. In D. Gabbay & F. Guenthner, editors: *Handbook of Philosophical Logic*, Kluwer Academic Publishers, pp. 1–128. Available at https://doi.org/10.1007/978-94-017-0452-6.

[23] Leandro Gomes, Alexandre Madeira & Luís Soares Barbosa (2019): *Generalising KAT to Verify Weighted Computations*. *Sci. Ann. Comput. Sci.* 29(2), pp. 141–184, doi:10.7561/SACS.2019.2.141.

[24] David Harel, Jerzy Tiuryn & Dexter Kozen (2000): *Dynamic Logic*. MIT Press, Cambridge, MA, USA, doi:10.7551/mitpress/2516.001.0001.

[25] John A. Kalman (1958): *Lattices with involution*. *Transactions of the American Mathematical Society* 87, pp. 485–491, doi:10.1090/S0002-9947-1958-0095135-X. Available at https://api.semanticscholar.org/CorpusID:53394259.

[26] S. C. Kleene (1956): *Representation of Events in Nerve Nets and Finite Automata*, pp. 3–42. Princeton University Press, Princeton. Available at https://doi.org/10.1515/9781400882618-002.

[27] D. Kozen (1992): *The Design and Analysis of Algorithms*. Monographs in Computer Science, Springer New York, NY. Available at https://doi.org/10.1007/978-1-4612-4400-4.

[28] D. Kozen (1994): *A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events*. *Information and Computation* 110(2), pp. 366–390, doi:10.1006/inco.1994.1037. Available at https://www.sciencedirect.com/science/article/pii/S0890540184710376.

[29] Dexter Kozen (1997): *Kleene Algebra with Tests*. *ACM Trans. Program. Lang. Syst.* 19(3), pp. 427–443, doi:10.1145/256167.256195.

[30] Dexter Kozen & Maria-Cristina Patron (2000): *Certification of Compiler Optimizations Using Kleene Algebra with Tests*. In John Lloyd, Veronica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv & Peter J. Stuckey, editors: *Computational Logic — CL 2000*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 568–582. Available at https://doi.org/10.1007/3-540-44957-4_38.

[31] Marcus Kracht (1998): *On Extensions of Intermediate Logics by Strong Negation*. *Journal of Philosophical Logic* 27(1), pp. 49–73, doi:10.1023/A:1004222213212.

[32] W. Kuich & A. Salomaa (1986): *Semirings, Automata, Languages*. EATCS monographs on theoretical computer science, Springer-Verlag. Available at https://doi.org/10.1007/978-3-642-69959-7.

[33] Vaughan Pratt (1990): *Dynamic algebras as a well-behaved fragment of relation algebras*. In Clifford H. Bergman, Roger D. Maddux & Don L. Pigozzi, editors: *Algebraic Logic and Universal Algebra in Computer Science*, Springer New York, New York, NY, pp. 77–110. Available at https://doi.org/10.1007/BFb0043079.

[34] John Preskill (2018): *Quantum Computing in the NISQ era and beyond*. *Quantum* 2, p. 79, doi:10.22331/q-2018-08-06-79.

[35] Graham Priest (2007): *Paraconsistency and dialetheism*. In Dov M. Gabbay & John Woods, editors: *The Many Valued and Nonmonotonic Turn in Logic*, *Handbook of the History of Logic* 8, Elsevier, pp. 129–204. Available at https://doi.org/10.1016/S1874-5857(07)80006-9.

[36] H. Rasiowa & R. Sikorski (1970): *The Mathematics of Metamathematics*. Monografie matematyczne, PWN-Polish Scientific Publishers. Available at https://books.google.pt/books?id=vtRGtQEACAAJ.

[37] Valentin N Redko (1964): *On defining relations for the algebra of regular events*. *Ukrainskii Matematicheskii Zhurnal* 16(1).

[38] Arto Salomaa (1966): *Two Complete Axiom Systems for the Algebra of Regular Events*. *J. ACM* 13(1), pp. 158–169, doi:10.1145/321312.321326.

[39] Alfred Tarski (1941): *On the Calculus of Relations*. *The Journal of Symbolic Logic* 6(3), pp. 73–89, doi:10.2307/2268577. Available at http://www.jstor.org/stable/2268577.

[40] Constantine Tsinakis & Annika M. Wille (2006): *Minimal Varieties of Involutive Residuated Lattices*. *Studia Logica* 83(1), pp. 407–423, doi:10.1007/s11225-006-8311-7.