Second International Workshop on Reconfigurable Transition Systems: Semantics, Logics and Applications (ReacTS 2025)

Preliminary proceedings
Toledo, Spain
November 11, 2025

(Please do not distribute)

Preface

This volume contains the papers presented at the 2nd International Workshop on Reconfigurable Transition Systems: Semantics, Logics and Applications, held in Toledo, Spain, on November 11, 2025.

This workshop aims to bring together the whole community of researchers working on different ways to model reconfigurable and reactive systems from a formal perspective. This includes theoretical approaches (like hybrid logics, reactive frames, model-update logics, and topological and algebraic semantics), or formalisms designed for specific purposes (like separation logic in software verification, dynamic epistemic logic in AI planning, and others). Also, our goal is to devise novel approaches and potential applications, and share a common perspective on the discipline.

We received 8 submissions from authors in 7 countries, out of which the Program Committee selected 4 to be accepted as regular papers and 4 to be accepted as short papers. All submitted papers were reviewed, on average, by three referees in a single blind manner. The conference pre-proceedings were made available at the conference. The proceedings includes all the final versions of the papers that took into account the comments received by the reviewers. All authors will be invited to submit their publications for the post-proceedings, to be published in a dedicated LNCS proceedings combining more workshops associated to the SEFM 2025 conference. Authors of selected accepted papers will be invited to submit extended versions of their contributions to appear in a special issue.

We would like to thank all researchers who submitted their work to the conference, to all colleagues who served on the Program Committee, and to the external reviewers, who helped us to prepare a high-quality conference program. Particular thanks to the invited speakers, Alessandra Palmigiano, from Vrije Universiteit Amsterdam in the Netherlands, and Amanda Vidal, from the Czech Academy of Sciences in Czech Republic, for their efforts and dedication to present their research and to share their perspectives on reconfigurable transitions systems at ReacTS. We are extremely grateful for the help in managing practical arrangements from the local organizers at the University of Castilla-La Mancha. We also thank Springer for their sponsorship.

November 2025 — Umberto Rivieccio
Toledo, Spain — José Proença

Organization

Program Chairs

Umberto Rivieccio	Universidad Nacional de Educación a Distancia, Spain
José Proença	CISTER & University of Porto, Portugal

Program Committee

Luís Soares Barbosa	Universidade Do Minho, Portugal
Benjamin Bedregal	Universidade Federal do Rio Grande do Norte, Brazil
Mario Benevides	Universidade Federal Fluminense, Brazil
Patrick Blackburn	University of Roskilde, Denmark
Valentin Cassano	Universidad Nacional de Río Cuarto and CONICET, Argentina
Madalena Chaves	Centre Inria d'Université Côte d'Azur, France
Gabriel Ciobanu	Institute of Computer Science, Romanian Academy, Romania
Raul Fervari	Universidad Nacional de Córdoba and CONICET, Argentina
Daniel Figueiredo	University of Aveiro, Portugal
Sabine Frittella	Université d'Orleans, France
Sujata Gosh	Indian Statistical Institute, Chennai, India
Andreas Herzig	CNRS, Université Paul Sabatier, France
Juha Kontinen	University of Helsinki, Finland
Alexandre Madeira	University of Aveiro, Portugal
Sérgio Marcelino	IT & Dep. Mathematics IST, University of Lisbon, Portugal
Vanina Martínez	IIIA-CSIC, Barcelona, Spain
Manuel Martins	University of Aveiro, Portugal
Carles Noguera i Clofent	Università degli Studi di Siena, Italy
Aybüke Özgün	University of Amsterdam, The Netherlands
Alessandra Palmigiano	Vrije Universiteit Amsterdam, The Netherlands
Regivan Santiago	Universidade Federal do Rio Grande do Norte, Brazil
François Schwarzentruber	ENS Rennes, France
Sonja Smets	ILLC, University of Amsterdam, The Netherlands

Ionuț Țuțu	Simion Stoilow Institute of Mathematics of the Romanian Academy, Romania
Fernando R. Velázquez-Quesada	University of Bergen, Norway
Fan Yang	Utrecht University, The Netherlands

Additional Reviewers

Vincent Hugot, INSA CVL, France
Maria João Frade, University of Minho, Portugal

Table of Contents

Regular Publications

Short Publications

Comparing finite many-valued logics via WSkS

Sérgio Marcelino[1 [0000−0002−6941−7555]]

SQIG - Instituto de Telecomunicações
Departamento de Matemática - Instituto Superior Técnico, Lisboa, Portugal
smarcel@math.tecnico.ulisboa.pt

Abstract. Partial non-deterministic logical matrices (PNmatrices), which interpret logical connectives as multi-functions, significantly enhance the expressive power of logical matrix semantics. This expressivity enables finite characterizations of broader classes of logics and facilitates solutions to compositionality problems. However, it also introduces challenges: notably, determining whether two finite PNmatrices characterize the same sets of theorems (Fmla-logics), or the same Tarskian consequence relations ($\mathsf{Set} \times \mathsf{Fmla}$-logics), is undecidable, even when restricted to languages with a finite number of variables.

We reduce the problem of comparing finite-variable fragments of multiple-conclusion consequence relations ($\mathsf{Set} \times \mathsf{Set}$-logics) of finite PNmatrices to the validity problem of Weak Second-order Logic of k Successors (WSkS), which is known to be decidable. Consequently, we provide an effective method for comparing the $\mathsf{Set} \times \mathsf{Set}$-logics of two finite PNmatrices, provided these are axiomatizable using a known finite number of variables.

Keywords: Many-valued logics · Partial non-deterministic matrices · Comparison of logics · Decidability · WSkS.

1 Introduction

The advantages of moving beyond truth-functional semantics for propositional logics have been increasingly noted in the literature. Non-deterministic interpretations of connectives – thus allowing the interpretation of connectives not to be completely determined by the input – was already considered in the 1980s to overcome the limitations of standard logical matrix semantics in finitely characterizing modal logics [20,19]. These ideas have garnered increasing attention over the past 20 years. The systematic study of *partial non-deterministic matrices* (PNmatrices) was initiated in [3,5], where their potential was first illustrated by producing finite many-valued semantics for a wide range of paraconsistent logics [2]. Subsequent work has revealed their broader expressive power: PNmatrix semantics can be automatically generated for certain families of sequent calculi [21,5], effectively updated to reflect axiomatic strengthenings with axioms of specific shapes [4,12,8], and composed in a modular fashion to yield semantics for *combined logics* [24,9]. This generalized semantical setting allows for the systematic study of the existence of axiomatizations, for a given logic, using rules

that avoid mixing certain connectives [10], or for greater control when extending a base logic with connectives satisfying certain desired properties [1,17].

As originally proposed in [20], PNmatrices can be used as finite-valued starting points for more involved semantics based on *level-valuations*, which recursively sieve out undesired valuations over a given finite PNmatrix. This idea has been increasingly used to give alternative semantics to modal logics that avoid possible worlds [13,27,28] and that still provide effective decision procedures for the characterized logics [22,16], leading to the notion of *restricted Nmatrices* (RNmatrices) [14].

In this paper, we tackle the problem of comparing the logics characterized by PNmatrices. The question about the computational status of this problem was first raised in [35] and has become increasingly relevant due to the growing number of finite PNmatrices in the literature, along with the rapid development of techniques for generating new ones. Notably, the ability to compare the logics of two PNmatrices is crucial for logical modeling tasks, such as identifying minimal semantics for a target logic. Moreover, it relates to the problem of obtaining 'local' characterizations of sound morphisms or quotients of Nmatrices, analogous to congruences in logical matrices [11].

What was not completely clear from the start is that the decidability of the problem may depend on the notion of logic one is working with. In [6] it is shown that if we take logic as a (single-conclusion) Tarskian consequence relation ($\mathsf{Set} \times \mathsf{Fmla}$-logic), this problem is undecidable. The present article demonstrates that when considering (multiple-conclusion) Scottian consequence relations ($\mathsf{Set} \times \mathsf{Set}$-logics), at least under finite variable restrictions, the problem becomes decidable. While the unrestricted problem remains open, the restricted version is enough to be usable when we know the considered logics are axiomatizable using a finite known number of variables.

These results align with known fundamental differences between the $\mathsf{Set} \times \mathsf{Set}$ and $\mathsf{Set} \times \mathsf{Fmla}$ settings. While the $\mathsf{Set} \times \mathsf{Set}$-logic of any finite matrix is finitely axiomatizable [31], this property fails for $\mathsf{Set} \times \mathsf{Fmla}$-logics [34]. Moreover, the $\mathsf{Set} \times \mathsf{Set}$ setting exhibits several distinctive features: it admits analytical axiomatizations [25,7], preserves finiteness under the operation of joining logics (unlike in $\mathsf{Set} \times \mathsf{Fmla}$) [9]; see [23] for an illustration of these differences in a concrete case of a connective that can only be fully captured in $\mathsf{Set} \times \mathsf{Set}$-context.

2 Preliminaries

This section provides an overview of the fundamental concepts related to PNmatrices and their associated logics. Due to space constraints, the discussion is not exhaustive; for a more comprehensive presentation, please refer to [9].

Propositional logics A propositional signature is an indexed family $\Sigma = \{\Sigma^{(k)} : k \in \mathbb{N}\}$ where $\Sigma^{(k)}$ is the set of k-ary connectives. We will be considering only finite signatures in which $\bigcup \Sigma$ is finite, and we denote by $\mathsf{maxArity}(\Sigma)$ the maximum k such that $\Sigma^{(k)} \neq \emptyset$.

We consider a set of propositional variables $P = \{p_i : i \in \mathbb{N}\}$, and given a set $Q \subseteq P$, we denote by $L_\Sigma(Q)$ the set of formulas written with connectives in Σ using variables in Q. The propositional language associated to Σ is $L_\Sigma(P)$, and its restriction to n variables is $L_\Sigma(P^{|n})$ with $P^{|n} = \{p_1, \ldots, p_n\}$. When Σ is fixed by the context, we simply write $Fm := L_\Sigma(P)$ and $Fm_n := L_\Sigma(P^{|n})$.

There are various notions of logic considered in the literature (see [30,31,18,9]):

- a Fmla-logic is just a set $\mathsf{Thm} \subseteq Fm$ of formulas closed for substitutions. This corresponds to the traditional notion of a logic defined by its set of valid formulas or theorems. In classical and intuitionistic settings, this notion is often sufficient, since the presence of an implication connective satisfying the Deduction Theorem allows the entire consequence relation to be recovered from its theorems.
- a $\mathsf{Set} \times \mathsf{Fmla}$-logic is a relation $\vdash \subseteq \wp(Fm) \times Fm$ satisfying reflexivity, monotonicity, transitivity, and substitution invariance. This captures the standard notion of logic as (*single-conclusion*) Tarskian consequence relation: a formula φ follows from a set of premises Γ (written $\Gamma \vdash \varphi$) whenever every formula in Γ is true, φ must be true.
- a $\mathsf{Set} \times \mathsf{Set}$-logic is a relation $\rhd \subseteq \wp(Fm) \times \wp(Fm)$ satisfying
  $\Gamma \rhd \Delta$ if $\Gamma \cap \Delta \neq \emptyset$ (*overlap*)
  $\Gamma \cup \Gamma' \rhd \Delta \cup \Delta'$ if $\Gamma \rhd \Delta$ (*dilution*)
  $\Gamma \rhd \Delta$ if $\Gamma \cup \Omega \rhd \overline{\Omega} \cup \Delta$ for every partition $\langle \Omega, \overline{\Omega} \rangle$ of L (*cut for sets*)
  $\Gamma^\sigma \rhd \Delta^\sigma$ for any $\sigma : P \to Fm$ if $\Gamma \rhd \Delta$ (*substitution invariance*).[1]
  This is the *multiple-conclusion* notion of consequence, in which both premises and conclusions are sets of formulas. Unlike the single-conclusion case, it is a *symmetric* notion: it can be read left-to-right as *truth-preserving* (if all formulas in Γ are true, then some formula in Δ is true), and right-to-left as *falsity-preserving* (if all formulas in Δ are false, then some formula in Γ is false).

We have adopted the notation (Fmla-, $\mathsf{Set} \times \mathsf{Fmla}$-, and $\mathsf{Set} \times \mathsf{Set}$-logics) introduced by Humberstone in [18]. The notion of $\mathsf{Set} \times \mathsf{Set}$-logic extends the ones of $\mathsf{Set} \times \mathsf{Fmla}$-logic and Fmla-logic, in the sense that $\vdash_\rhd := \{\langle \Gamma, \varphi \rangle : \Gamma \rhd \{\varphi\}\}$ is a $\mathsf{Set} \times \mathsf{Fmla}$-logic, and $\mathsf{Thm}_\rhd := \{\varphi : \emptyset \rhd \{\varphi\}\}$ is a Fmla-logic.

We say that $\rhd$ is *compact* whenever $\Gamma \rhd \Delta$ implies $\Gamma' \rhd \Delta'$ for some finite $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$.

The n-variable fragment of each notion of logic is denoted by

$$\mathsf{Thm}^{|n} := \mathsf{Thm} \cap Fm_n \quad \vdash^{|n} := \vdash \cap\, \wp(Fm_n) \times Fm_n \quad \rhd^{|n} := \rhd \cap\, \wp(Fm_n) \times \wp(Fm_n)$$

PNmatrix semantics A *Σ-partial non-deterministic matrix* (Σ-PNmatrix) is a tuple $\mathbb{M} = \langle A, \cdot_\mathbb{M}, D \rangle$, where A is a set of truth-values, $\cdot_\mathbb{M}$ interprets each i-ary connective $f \in \Sigma$ as a multi-function $f_\mathbb{M} : A^i \to \wp(A)$, and $D \subseteq A$ is the set of *designated* truth-values. We say that $\mathbb{M} = \langle A, \cdot_\mathbb{M}, D \rangle$ is *finite* whenever Σ and

[1] Γ^σ is obtained from Γ by replacing each variable with its image under σ.

A are finite, and set $\mathsf{size}(\mathbb{M}) = |A|$. We will often drop the Σ- prefix when Σ is fixed by the context.

We say $\mathbb{M}$ is a *non-deterministic matrix* (Nmatrix) whenever $f_{\mathbb{M}}(\boldsymbol{a}) \neq \emptyset$ for every $f \in \Sigma^{(i)}$ and $\boldsymbol{a} \in A^i$, and that $\mathbb{M}$ is a *matrix* whenever $f_{\mathbb{M}}(\boldsymbol{a})$ is a singleton for every $f \in \Sigma^{(i)}$ and $\boldsymbol{a} \in A^i$.

A valuation over a PNmatrix $\mathbb{M}$ is a function $v : Fm \to A$ that satisfies the condition: $v(f(\varphi_1, \ldots, \varphi_n)) \in f_{\mathbb{M}}(v(\varphi_1), \ldots, v(\varphi_n))$, for every connective $f \in \Sigma$ and formulas $\varphi_1, \ldots, \varphi_n \in Fm$. The set of all valuations over $\mathbb{M}$ is denoted by $\mathsf{Val}(\mathbb{M})$.

The $\mathsf{Set} \times \mathsf{Set}$-logic $\rhd_{\mathbb{M}}$ characterized by $\mathbb{M}$ is given by

$$\Gamma \rhd_{\mathbb{M}} \Delta \text{ whenever for every } v \in \mathsf{Val}(\mathbb{M}), \text{ if } v(\Gamma) \subseteq D \text{ then } v(\Delta) \cap D \neq \emptyset.$$

The Fmla-logic characterized by $\mathbb{M}$ is $\mathsf{Thm}(\mathbb{M}) := \mathsf{Thm}_{\rhd_{\mathbb{M}}}$, and the $\mathsf{Set} \times \mathsf{Fmla}$-logic characterized by $\mathbb{M}$ is $\vdash_{\mathbb{M}} := \vdash_{\rhd_{\mathbb{M}}}$.

Given PNmatrix $\mathbb{M} = \langle A, \cdot_{\mathbb{M}}, D\rangle$ and $X \subseteq A$, we denote by $\mathbb{M}_X$ the restriction of $\mathbb{M}$ to X. We let $\mathcal{T}_{\mathbb{M}} := \{X \subseteq A : \mathbb{M}_X \text{ is Nmatrix}\}$ be the set of *total components* of $\mathbb{M}$, and $\mathcal{T}^*_{\mathbb{M}} := \bigcup_{X \in \mathcal{T}_{\mathbb{M}}} \wp(X)$.

Fixed Υ closed for subformulas ($\Upsilon = \mathsf{sub}(\Upsilon)$) we let the set $\mathsf{Val}_\Upsilon(\mathbb{M}) := \{v_{|\Upsilon} : v \in \mathsf{Val}(\mathbb{M})\}$ where $v_{|\Upsilon}$ denotes the restriction of v to Υ. We have that $\mathbb{v} \in \mathsf{Val}_\Upsilon(\mathbb{M})$ iff both conditions in lines (1) and (2) hold

$$\mathbb{v}(f(\varphi_1, \ldots, \varphi_i)) \in f_{\mathbb{M}}(\mathbb{v}(\varphi_1), \ldots, \mathbb{v}(\varphi_i)) \text{ for every } f(\varphi_1, \ldots, \varphi_i) \in \Upsilon, \tag{1}$$

$$\mathbb{v}(\Upsilon) \in \mathcal{T}^*_{\mathbb{M}}. \tag{2}$$

Partial bivaluations Each partial valuation $\mathbb{v} \in \mathsf{Val}_\Upsilon(\mathbb{M})$ induces a partial *bivaluation* $\mathbb{b}_{\mathbb{v}} : Fm \to \{0, 1\}$ as:

$$\mathbb{b}_{\mathbb{v}}(\varphi) = \begin{cases} 1 & \text{if } \mathbb{v}(\varphi) \in D, \\ 0 & \text{otherwise.} \end{cases}$$

The set of all Υ-partial bivaluations on $\mathbb{M}$ is $\mathsf{Bival}_\Upsilon(\mathbb{M}) = \{\mathbb{b}_{\mathbb{v}} : \mathbb{v} \in \mathsf{Val}_\Upsilon(\mathbb{M})\}$.

Lemma 1. *For any finite PNmatrices $\mathbb{M}_1$ and $\mathbb{M}_2$, $\rhd^{|n}_{\mathbb{M}_1} \subseteq \rhd^{|n}_{\mathbb{M}_2}$ iff $\mathsf{Bival}_\Upsilon(\mathbb{M}_2) \subseteq \mathsf{Bival}_\Upsilon(\mathbb{M}_1)$ for all finite $\Upsilon = \mathsf{sub}(\Upsilon) \subseteq Fm_n$.*

Proof. Since $\mathbb{M}_1$ and $\mathbb{M}_2$ are finite, $\rhd_1$ and $\rhd_2$ are compact. Thus, $\rhd^{|n}_1 \subseteq \rhd^{|n}_2$ iff $\Gamma \rhd_1 \Delta$ implies $\Gamma \rhd_2 \Delta$ for any finite $\Gamma, \Delta \subseteq Fm_n$.

Let $\Upsilon = \mathsf{sub}(\Gamma \cup \Delta)$ and $i \in \{1, 2\}$. We have that $\Gamma \rhd_i \Delta$ iff there is no $\mathbb{v}_i \in \mathsf{Val}(\mathbb{M}_i)_\Upsilon$ such that $\mathbb{v}_i(\Gamma) \subseteq D$ and $\mathbb{v}_i(\Delta) \cap D = \emptyset$. Equivalently, $\Gamma \rhd_i \Delta$ iff there is no $\mathbb{b}_i \in \mathsf{Bival}(\mathbb{M}_i)_\Upsilon$ such that $\mathbb{b}_i(\Gamma) = 1$ and $\mathbb{b}_i(\Delta) = 0$.

Further, note that for $\mathbb{b} : \Upsilon \to \{0, 1\}$, if $\mathbb{b} \in \mathsf{Bival}_\Upsilon(\mathbb{M}_2) \setminus \mathsf{Bival}_\Upsilon(\mathbb{M}_1)$ we have that $\mathbb{b}^{-1}(1) \not\rhd_{\mathbb{M}_2} \mathbb{b}^{-1}(0)$ and $\mathbb{b}^{-1}(1) \rhd_{\mathbb{M}_1} \mathbb{b}^{-1}(0)$, and thus $\rhd^{|n}_1 \not\subseteq \rhd^{|n}_2$.

Hence, $\rhd^{|n}_1 \subseteq \rhd^{|n}_2$ iff $\mathsf{Bival}_\Upsilon(\mathbb{M}_2) \subseteq \mathsf{Bival}_\Upsilon(\mathbb{M}_1)$ for every finite $\Upsilon = \mathsf{sub}(\Upsilon) \subseteq Fm_n$. □

3 Computational problems

Given a class $\mathcal{M}$ of finite PNmatrices, we are interested in the following computational problems where $n \in \mathbb{N} \cup \{\omega\}$:

$\mathsf{Thm}^{|n}_{\subseteq}(\mathcal{M})$: given a finite signature Σ and finite Σ-PNmatrices $\mathbb{M}_1, \mathbb{M}_2 \in \mathcal{M}$ determine whether $\mathsf{Thm}(\mathbb{M}_1)^{|n} \subseteq \mathsf{Thm}(\mathbb{M}_2)^{|n}$.
$\mathsf{Sing}^{|n}_{\subseteq}(\mathcal{M})$: given a finite signature Σ and finite Σ-PNmatrices $\mathbb{M}_1, \mathbb{M}_2 \in \mathcal{M}$ determine whether $\vdash^{|n}_{\mathbb{M}_1} \subseteq \vdash^{|n}_{\mathbb{M}_2}$.
$\mathsf{Mult}^{|n}_{\subseteq}(\mathcal{M})$: given a finite signature Σ and finite Σ-PNmatrices $\mathbb{M}_1, \mathbb{M}_2 \in \mathcal{M}$ determine whether $\rhd^{|n}_{\mathbb{M}_1} \subseteq \rhd^{|n}_{\mathbb{M}_2}$.

Consider the following classes of PNmatrices: PNmatr is the class of all finite PNmatrices, Nmatr is the class of all finite Nmatrices, Matr is the class of all finite matrices. Clearly, $\mathsf{Matr} \subsetneq \mathsf{Nmatr} \subsetneq \mathsf{PNmatr}$.

A problem is *decidable* if there exists an algorithm that, for each suitable input, provides a yes/no answer depending on whether the condition holds. If a problem is decidable with respect to a class $\mathcal{M}$, it is also decidable with respect to any subclass $\mathcal{M}' \subseteq \mathcal{M}$. As usual, the dual of a computational problem is the computational problem obtained by negating the envisaged condition. Given a computational problem P we will denote its *dual* by $\overline{\mathsf{P}}$. Recall from computability theory (e.g., [32]) that a problem P is decidable if and only if $\overline{\mathsf{P}}$ is decidable. Additionally, the notion of *computable reduction* (or many-one reduction), denoted by $\leq$, ensures that if $\mathsf{P} \leq \mathsf{Q}$ and Q is decidable, then P is also decidable.

In [26,6] the following problems were considered

$\exists\mathsf{Thm}(\mathcal{M})$: given a finite signature Σ and a finite Σ-Nmatrix $\mathbb{M} \in \mathcal{M}$ determine whether $\mathsf{Thm}(\mathbb{M}) \neq \emptyset$;
$\mathsf{Eqv}(\mathcal{M})$: given a finite signature Σ and finite Σ-Nmatrices $\mathbb{M}_1, \mathbb{M}_2 \in \mathcal{M}$ determine whether $\vdash_{\mathbb{M}_1} = \vdash_{\mathbb{M}_2}$.

Since the set of expressible functions of a given arity is computable in a finite algebra, the problems $\exists\mathsf{Thm}(\mathsf{Matr})$ and $\mathsf{Eqv}(\mathsf{Matr})$ are decidable (see [26,6]). Similarly, the problems $\mathsf{Thm}^{|n}_{\subseteq}(\mathsf{Matr})$ and $\mathsf{Sing}^{|n}_{\subseteq}(\mathsf{Matr})$ are decidable, for every $n \in \mathbb{N} \cup \omega$. In the presence of non-determinism the situation is radically different as $\exists\mathsf{Thm}(\mathsf{Nmatr})$ and $\mathsf{Eqv}(\mathsf{Nmatr})$ are undecidable (see [26,6]). The following proposition shows that undecidability propagates to $\mathsf{Thm}^{|\omega}_{\subseteq}(\mathsf{PNmatr})$ and $\mathsf{Sing}^{|\omega}_{\subseteq}(\mathsf{PNmatr})$, as to their finite variable variants.

Proposition 1. *For every* $0 < n \in \mathbb{N} \cup \{\omega\}$*:*

(i) The problem $\mathsf{Thm}^{|n}_{\subseteq}(\mathsf{PNmatr})$ *is undecidable.*
(ii) The problem $\mathsf{Sing}^{|n}_{\subseteq}(\mathsf{PNmatr})$ *is undecidable.*

Proof. Items (i) and (ii) are straightforward variations of the results in [6], and we provide only the most important details of the proof.

Given finite Σ-Nmatrix, consider $\mathbb{M}_{\mathsf{un}} = \langle \{0,1\}, \cdot_{\mathsf{un}}, \{1\}\rangle$ the 2-valued Σ-Nmatrix, where the connective interpretation is completely unconstrained, i.e, $f_{\mathbb{M}}(\boldsymbol{a}) = \{0,1\}$ for every $f \in \Sigma^{(i)}$ and $\boldsymbol{a} \in \{0,1\}^i$. Clearly, $\mathsf{Thm}(\mathbb{M}) = \mathsf{Thm}^{|n}(\mathbb{M}) = \emptyset$ for every $n \in \mathbb{N}\cup\{\omega\}$. As for every $n > 0$ we have $\mathsf{Thm}(\mathbb{M}) \neq \emptyset$ iff $\mathsf{Thm}^{|n}(\mathbb{M}) \neq \emptyset$, we obtain that $\mathsf{Thm}(\mathbb{M}) \neq \emptyset$ if and only if $\mathsf{Thm}^{|n}(\mathbb{M}) \not\subseteq \mathsf{Thm}^{|n}(\mathbb{M}_{\mathsf{un}})$ for each $n > 0$. Since $\mathbb{M}_{\mathsf{un}}$ is finite we have that $\exists\mathsf{Thm}(\mathsf{Nmatr}) \leq \overline{\mathsf{Thm}^{|n}_{\subseteq}(\mathsf{PNmatr})}$. As $\exists\mathsf{Thm}(\mathsf{Nmatr})$ is undecidable, so is $\mathsf{Thm}^{|n}_{\subseteq}(\mathsf{PNmatr})$ and (i) holds.

For (ii), consider the *tildeing*-operation on Nmatrices introduced in [6, Definition 14]. We have that $\widetilde{\mathbb{M}}$ is finite whenever $\mathbb{M}$ is. It was shown in [6] that $\Gamma \vdash_{\widetilde{\mathbb{M}}} \varphi$ iff $\varphi \in \Gamma$ or $\varphi \in \mathsf{Thm}_{\mathbb{M}}$. This implies that $\mathsf{Thm}(\mathbb{M}) \neq \emptyset$ iff $\vdash^{|n}_{\widetilde{\mathbb{M}}} \not\subseteq \vdash^{|n}_{\mathbb{M}_{\mathsf{un}}}$. Hence, $\exists\mathsf{Thm}(\mathsf{Nmatr}) \leq \overline{\mathsf{Sing}^{|n}_{\subseteq}(\mathsf{Nmatr})}$. So, $\mathsf{Sing}^{|n}_{\subseteq}(\mathsf{PNmatr})$ is undecidable since $\exists\mathsf{Thm}(\mathsf{Nmatr})$ is, and (ii) holds. □

The following theorem is the main result of the paper, that deepens the contrast on the computational status of the considered problem for finite PNmatrices depending on the notion of logic considered. We use WSkS (Weak Second-order Logic of k Successors) as a tool to establish the decidability of $\mathsf{Mult}^{|n}_{\subseteq}(\mathsf{PNmatr})$ for $n \in \mathbb{N}$. We present here only the essential definitions needed for this work, for more details see [29,33,15].

For $k \in \mathbb{N}$ let $\mathsf{T}_k = \{1,\ldots,k\}^*$ be the finite sequences of elements of $\{1,\ldots,k\}$. We identify T_k with the infinite full k-ary tree.

Given a set of first-order variables vars, the set of WSkS-terms is

$$\mathsf{Terms} = \mathsf{T}_k \cup \{x\tau : x \in \mathsf{vars}, \tau \in \mathsf{T}_k\},$$

where successors are denoted using postfix notation, $\mathsf{Suc}_i(\tau) = \tau i$ is the i successor of τ for $\tau \in \mathsf{T}_k$ and $1 \leq i \leq k$. The set WSkS formulas Fm_{WSkS} is built from atomic formulas of the form $\mathsf{Q}(t)$ for unary predicate $\mathsf{Q} \in \mathsf{Preds}$ and term $t \in \mathsf{Terms}$, and complex formulas are built from atomic formulas by the application of classical propositional connectives $\rightarrow$, $\wedge$, $\vee$, $\neg$, and first-order quantification over vars and second-order quantification on the unary predicates in Preds. The semantics of WSkS are given by interpretations $\mathbb{I}$ mapping first-order variables $x \in \mathsf{vars}$ to $x_{\mathbb{I}} \in \mathsf{T}_k$, and predicates $\mathsf{Q} \in \mathsf{Preds}$ to finite sets $\mathsf{Q}_{\mathbb{I}} \subseteq \mathsf{T}_k$. The interpretation $\mathbb{I}$ is extended to arbitrary terms by $\tau_{\mathbb{I}} = \tau$ and $(x\tau)_{\mathbb{I}} = x_{\mathbb{I}}\tau$, and we write

$$\mathbb{I} \models \mathsf{Q}(t) \quad \text{if} \quad t_{\mathbb{I}} \in \mathsf{Q}_{\mathbb{I}} \qquad \text{for} \qquad \mathsf{Q} \in \mathsf{Preds}$$

$$\mathbb{I} \models \Phi \rightarrow \Psi \quad \text{if} \quad \mathbb{I} \not\models \Phi \quad \text{or} \quad \mathbb{I} \models \Psi \quad \text{(likewise for } \wedge, \vee, \neg\text{)}$$

$$\mathbb{I} \models \forall x.\, \Phi \quad \text{if} \quad \mathbb{I}[x \mapsto \tau] \models \Phi \text{ for every } \quad \tau \in \mathsf{T}_k$$

$$\mathbb{I} \models \forall \mathsf{Q}.\, \Phi \quad \text{if} \quad \mathbb{I}[\mathsf{Q} \mapsto Q] \models \Phi \text{ for every finite } \quad Q \subseteq \mathsf{T}_k.$$

We say that a WSkS formula Φ is *valid* whenever $\mathbb{I} \models \Phi$ for every interpretation $\mathbb{I}$. Let $\mathsf{Valid}_{\mathrm{WSkS}}$ be the *validity problem for WSkS*: $\mathsf{Valid}_{\mathrm{WSkS}}(\Phi)$ outputs yes if Φ

is valid, and no otherwise. The decidability of $\mathsf{Valid}_{\mathrm{WSkS}}$ is a fundamental result in theoretical computer science, originating from Rabin's seminal work [29] and extensively studied since then [33,15].

Theorem 1. *For every $n \in \mathbb{N}$, the problem $\mathsf{Mult}^{|n}_{\subseteq}(\mathsf{PNmatr})$ is decidable.*

Proof. The proof relies on establishing a reduction $\mathsf{Mult}^{|n}_{\subseteq}(\mathsf{PNmatr}) \leq \mathsf{Valid}_{\mathrm{WSkS}}$, which results from Proposition 2. The proof of the reduction will be the focus of the next section. □

The next section is dedicated to constructing $\Phi^n_{\mathbb{M}_1\mathbb{M}_2}$, a WSkS formula that is valid iff $\rhd^{|n}_{\mathbb{M}_1} \subseteq \rhd^{|n}_{\mathbb{M}_2}$. Our construction builds upon Lemma 1, encoding the set of partial bivaluations for each finite PNmatrix using WSkS formulas. This approach is specific to the $\mathsf{Set} \times \mathsf{Set}$-context, as the analogues of Lemma 1 do not hold for Fmla and $\mathsf{Set} \times \mathsf{Fmla}$-logics.

Before proceeding, recall that $\rhd_R$ denotes the smallest $\mathsf{Set} \times \mathsf{Set}$-logic containing $R \subseteq \wp(Fm) \times \wp(Fm)$, and that R *axiomatizes* (or serves as a *basis for*) $\rhd_R$. We say that $\rhd_R$ is *axiomatized in n variables* whenever $R \subseteq \wp(Fm_n) \times \wp(Fm_n)$. This restriction to finitely many variables is sufficient when the logics under consideration are axiomatizable with a number of variables that can be effectively computed from the corresponding PNmatrix. In fact, since $\rhd_R \subseteq \rhd$ if and only if $R \subseteq \rhd$, it follows that if $\rhd_1$ is axiomatizable in n variables then $\rhd_1 \subseteq \rhd_2$ iff $\rhd^{|n}_1 \subseteq \rhd^{|n}_2$. Consequently, if $\rhd_{\mathbb{M}_1}$ is axiomatizable in n variables, then deciding whether $\rhd_{\mathbb{M}_1} \subseteq \rhd_{\mathbb{M}_2}$ reduces to checking the validity of $\Phi^n_{\mathbb{M}_1\mathbb{M}_2}$.

4 Reducing $\mathsf{Mult}^{|n}_{\subseteq}(\mathsf{PNmatr})$ to validity in WSkS

For readability, we denote sequences of predicates as $\mathbf{Q} := \mathsf{Q}_1 \dots \mathsf{Q}_\ell$ using vector notation. We write $\Phi(\mathbf{Q}, x_1, \dots, x_j)$ when the free second-order variables of Φ are listed in $\mathbf{Q}$, and the first-order free variables in $x_1, \dots, x_j$. Additionally, we use $\forall\mathbf{Q}.(\Phi)$ instead of $\forall\mathsf{Q}_1 \dots \forall\mathsf{Q}_k.(\Phi)$.

Let

$$\mathsf{Union}(\mathbf{Q}, x) := \bigvee_{1 \leq i \leq \ell} \mathsf{Q}_i(x),$$

$$\mathsf{AtMostOne}(\mathbf{Q}) := \forall x.\big(\bigwedge_{1 \leq i < j \leq \ell} \neg(\mathsf{Q}_i(x) \wedge \mathsf{Q}_j(x))\big).$$

It is immediate that $\mathbb{I} \models \mathsf{Union}(\mathbf{Q}, x)$ iff $x_{\mathbb{I}} \in \bigcup_{1 \leq i \leq \ell} (\mathsf{Q}_i)_{\mathbb{I}}$.

Furthermore, $\mathbb{I} \models \mathsf{AtMostOne}(\mathbf{Q})$ iff for every $\tau \in \mathsf{T}_k$, τ belongs to at most one $(\mathsf{Q}_i)_{\mathbb{I}}$ for $1 \leq i \leq \ell$.

Given $\Phi(x)$ with free variable x, we define:

$$\mathsf{ClosedForPrefix}(\Phi) := \forall x.\big((\bigvee_{1 \leq i \leq k} \Phi(xi)) \to \Phi(x)\big).$$

We have that $\mathbb{I} \models \mathsf{ClosedForPrefix}(\Phi)$ iff $\Phi_{\mathbb{I}} := \{x : \mathbb{I} \models \Phi\}$ is closed for prefixes. Let $\mathsf{Labels}(\mathbf{Q}) := \mathsf{AtMostOne}(\mathsf{Q}) \wedge \mathsf{ClosedForPrefix}(\mathsf{Union}(\mathsf{Q}, x))$.

When $\mathbb{I} \models \mathsf{Labels}(\mathbf{Q})$, the interpretation $\mathbb{I}$ partitions $\mathsf{T}_{\mathbb{I}} := \{\tau \in \mathsf{T}_k : \mathbb{I} \models \mathsf{Union}(\mathsf{Q}, \tau)\}$ into ℓ disjoint subsets, each labelled by some Q_i that occurs in $\mathbf{Q}$. Furthermore, $\mathsf{T}_{\mathbb{I}}$ is a finite subtree of T_k rooted at ϵ. This subtree is referred to as a *finite prefix subtree* of T_k.

Encoding sets of formulas in $\boldsymbol{Fm_n}$ We consider fixed $n \in \mathbb{N}$ and finite signature Σ, let $k := \max(2, \mathsf{maxArity}(\Sigma))$, and consider the following predicates dedicated to sets of formulas: a predicate A (labelling *append nodes*); predicates C_f for $f \in \Sigma$, and P_i for $p_i \in P^{|n}$ (labelling *formula nodes*).

Letting $\bigcup \Sigma = \{f_1, \ldots, f_m\}$ and $\Sigma^{(i)} = \{f_{i,1}, \ldots, f_{i,m_i}\}$ we consider

$$\mathbf{C} := \mathsf{C}_{f_1} \ldots \mathsf{C}_{f_m} \qquad \mathbf{C}_{=i} := \mathsf{C}_{f_{i,1}} \ldots \mathsf{C}_{f_{i,m_i}} \qquad \mathbf{P} := \mathsf{P}_1 \ldots \mathsf{P}_n$$

and refer to larger lists of predicates by affixation, letting $\mathbf{S} := \mathsf{A}\mathbf{C}\mathbf{P}$ list every predicate dealing with sets formulas, $\mathbf{F} := \mathbf{C}\mathbf{P}$ list every predicate dealing with formulas. We further let

$$\mathsf{IsRelevant}(\mathbf{S}, x) := \mathsf{Union}(\mathbf{S}, x) \qquad \mathsf{IsForm}(\mathbf{S}, x) := \mathsf{Union}(\mathbf{F}, x)$$
$$\mathsf{IsConnAr}_i(\mathbf{S}, x) := \mathsf{Union}(\mathbf{C}_{=i}, x) \qquad \mathsf{IsVar}(\mathbf{S}, x) := \mathsf{Union}(\mathbf{P}, x)$$

$$\mathsf{IsSetOfForm}(\mathbf{S}) := \mathsf{Labels}(\mathbf{S}) \wedge \mathsf{TopLeft}(\mathbf{S}) \wedge \mathsf{BotRight}(\mathbf{S})$$

$$\mathsf{TopLeft}(\mathbf{S}) := \mathsf{A}(\epsilon) \wedge \forall x. \Big(\mathsf{A}(x) \rightarrow \big((\mathsf{IsRelevant}(\mathbf{S}, x1) \rightarrow \mathsf{A}(x1)) \wedge \tag{3}$$
$$\mathsf{IsForm}(\mathbf{S}, x2) \wedge \tag{4}$$
$$\bigwedge_{3 \leq i \leq k} \neg \mathsf{IsRelevant}(\mathbf{S}, xi) \big) \Big) \tag{5}$$

$$\mathsf{BotRight}(\mathbf{S}) := \bigwedge_{0 \leq i \leq n} \Big(\mathsf{IsConnAr}_i(\mathbf{S}, x) \rightarrow$$
$$\big(\bigwedge_{1 \leq j \leq i} \mathsf{IsForm}(\mathbf{S}, xj) \wedge \bigwedge_{i < j \leq k} \neg \mathsf{IsRelevant}(\mathbf{S}, xj) \big) \Big) \wedge \tag{6}$$
$$\big(\mathsf{IsVar}(\mathbf{S}, x) \rightarrow \bigwedge_{1 \leq j \leq k} \neg \mathsf{IsRelevant}(\mathbf{S}, xj) \big) \tag{7}$$

Assume that $\mathbb{I} \models \mathsf{IsSetOfForm}(\mathbf{S})$. From $\mathbb{I} \models \mathsf{Labels}(\mathbf{S})$ we have that $\mathbb{I}$ labels with predicates in $\mathbf{S}$ exactly the finite tree $\mathsf{T}_{\mathbb{I}} = \{\tau \in \mathsf{T}_k : \mathsf{IsRelevant}(\mathbf{S}, \tau)\}$ illustrated in Fig. 1. From $\mathbb{I} \models \mathsf{TopLeft}(\mathbf{S})$ we have that append nodes are in the top left corner by line (3), and their 2-successors are always formula nodes by line (4). Further, append nodes have maximum branching of 2 by line (5). Formula nodes are below them on the right by line (4), as depicted in Fig. 2.

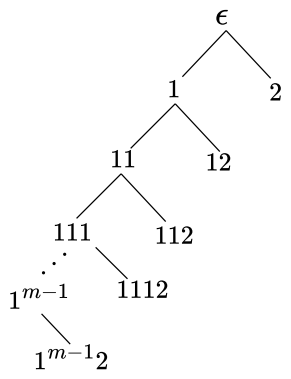


Fig. 1: The finite prefix subtree $\mathsf{T}_{\mathbb{I}}$.

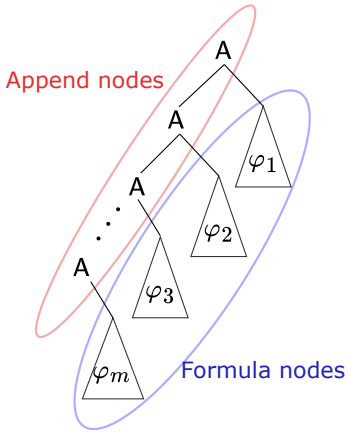


Fig. 2: Append and formula nodes.

From $\mathbb{I} \models \mathsf{BotRight}(\mathbf{S})$, using line (6) we have that formula nodes labelled by predicates occurring in $\mathsf{C}_{=i}$ are i-branching, corresponding to i-ary connectives $f \in \Sigma^{(i)}$. By line (7) nodes labelled by predicates in $\mathbf{P}$, corresponding to variable p_i for some $1 \leq i \leq n$, have no successors in $\mathsf{T}_{\mathbb{I}}$. Leaves of $\mathsf{T}_{\mathbb{I}}$ are labelled by predicates listed in $\mathbf{C}_{=0}\mathbf{P}$. This encoding captures the usual tree representation of formulas as illustrated in Fig. 3.

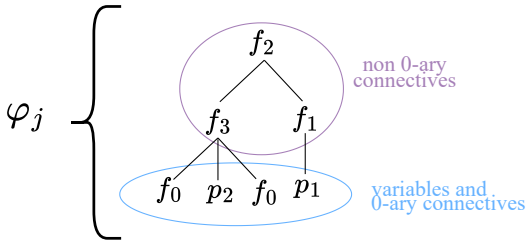


Fig. 3: Tree representation of $\varphi_j = f_2(f_3(f_0, p_2, f_0), p_1)$ assuming $f_i \in \Sigma^{(i)}$ for $i = 0, 1, 2, 3$.

Still assuming $\mathbb{I} \models \mathsf{IsSetOfForm}(\mathbf{S})$ we consider

$$\Upsilon_{\mathbb{I}} = \{\varphi_x^{\mathbb{I}} : \mathsf{IsForm}(\mathbf{S}, x)\} \text{ where } \varphi_x^{\mathbb{I}} = \begin{cases} f(\varphi_{x1}^{\mathbb{I}}, \ldots, \varphi_{xi}^{\mathbb{I}}) & \text{if } \mathsf{C}_f(x), \\ p_j & \text{if } \mathsf{P}_j(x). \end{cases}$$

We have that $\Upsilon_{\mathbb{I}} \subseteq Fm_n$ is closed for subformulas, as leaf nodes are labelled by 0-ary connectives and variables, each non-leaf formula node is labelled by a connective with arity corresponding to its branching relative to $\mathsf{T}_{\mathbb{I}}$, and its immediate successors encode the maximal strict subformulas. Indeed, the $\mathbf{S}$-labelled subtree of $\mathsf{T}_{\mathbb{I}}$ below $x_{\mathbb{I}}$ is the usual tree representation of $\varphi_x^{\mathbb{I}}$.
We can express that two nodes $x, y \in \mathsf{T}_{\mathbb{I}}$ represent the same formula recursively checking the equality between the labels of the matching nodes along the subtree

of $\mathsf{T}_{\mathbb{I}}$ below each x and y:

$$\begin{aligned}
&\mathsf{SameForm}(\mathbf{S}, x, y) := \\
&\qquad \mathsf{IsForm}(\mathbf{S}, x) \wedge \mathsf{IsForm}(\mathbf{S}, y) \wedge && (8)\\
&\qquad \bigwedge_{f \in \bigcup \Sigma} (\mathsf{C}_f(x) \leftrightarrow \mathsf{C}_f(y)) \wedge \bigwedge_{1 \leq i \leq n} (\mathsf{P}_i(x) \leftrightarrow \mathsf{P}_i(y)) \wedge && (9)\\
&\qquad \bigwedge_{1 \leq i \leq k} \big((\mathsf{IsForm}(\mathbf{S}, xi) \vee \mathsf{IsForm}(\mathbf{S}, yi)) \rightarrow \mathsf{SameForm}(\mathbf{S}, xi, yi)\big) && (10)
\end{aligned}$$

If $\mathbb{I} \models \mathsf{IsSetOfForm}(\mathbf{S}) \wedge \mathsf{SameForm}(\mathbf{S}, x, y)$, then both $x_{\mathbb{I}}$ and $y_{\mathbb{I}}$ must be formula nodes by line (8), labelled by the same predicate listed in **CP** by line (9). In the base case, $x_{\mathbb{I}}$ and $y_{\mathbb{I}}$ are leaves of $\mathsf{T}_{\mathbb{I}}$ and have the same label in $\mathbf{C}_{=0}\mathbf{P}$. In case both $x_{\mathbb{I}}$ and $y_{\mathbb{I}}$ are labelled with the same predicate C in $\mathbf{C}_{=i}$ for $i > 0$, their successors must satisfy the recursive condition in line (10), ensuring that corresponding successors are either both non-formula nodes or represent the same formula. This enforces that $\varphi^{\mathbb{I}}_x = \varphi^{\mathbb{I}}_y$.

Valuations Given a finite Σ-Nmatrix $\mathbb{M} = \langle A, \cdot_{\mathbb{M}}, D\rangle$, we consider the predicates V_a for $a \in A$, write $\mathbf{V}_{\mathbb{M}} = \mathsf{V}_1, \ldots, \mathsf{V}_{|A|}$ and $\mathsf{HasValue}(\mathbf{V}_{\mathbb{M}}, x) = \mathsf{Union}(\mathbf{V}_{\mathbb{M}}, x)$ and

$$\begin{aligned}
&\mathsf{IsPartialValuation}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}}) := \\
&\qquad \forall x \forall y \Big(\big(\mathsf{IsForm}(\mathbf{S}, x) \rightarrow \mathsf{HasValue}(\mathbf{V}_{\mathbb{M}}, x)\big) \wedge && (11)\\
&\qquad\quad \mathsf{AtMostOne}(\mathbf{V}_{\mathbb{M}}, x) \wedge && (12)\\
&\qquad\quad \big(\mathsf{SameForm}(\mathbf{S}, x, y) \rightarrow \mathsf{SameValue}_{\mathbb{M}}(\mathbf{V}_{\mathbb{M}}, x, y)\big) \wedge && (13)\\
&\qquad\quad \mathsf{Respects}_{\mathbb{M}}(\mathbf{V}_{\mathbb{M}}, x) \wedge && (14)\\
&\qquad\quad \mathsf{Total}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}}) \Big) && (15)
\end{aligned}$$

where

$$\begin{aligned}
\mathsf{SameValue}_{\mathbb{M}}(\mathbf{V}_{\mathbb{M}}, x, y) &:= \bigwedge_{a \in A} (\mathsf{V}_a(x) \leftrightarrow \mathsf{V}_a(y)) && (16)\\
\mathsf{Respects}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}}, x) &:= \bigwedge_{f \in \bigcup \Sigma} \mathsf{Respects}_f(\mathbf{S}, \mathbf{V}_{\mathbb{M}}, x) && (17)\\
\mathsf{Respects}_f(\mathbf{S}, \mathbf{V}_{\mathbb{M}}, x) &:= \forall x \big((\mathsf{C}_f(x) \wedge \bigwedge_{1 \leq j \leq i} \mathsf{V}_{a_j}(xj)) \rightarrow \bigvee_{a \in \copyright_{\mathbb{M}}(\boldsymbol{a})} \mathsf{V}_a(x)\big) && (18)\\
&\qquad\qquad \text{for } \boldsymbol{a} = a_1, \ldots, a_i \text{ and } f \in \Sigma^{(i)}\\
\mathsf{Total}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}}) &:= \bigvee_{X \in \mathcal{T}_{\mathbb{M}}} \forall x \big(\bigvee_{a \in X} \mathsf{V}_a(x)\big) && (19)
\end{aligned}$$

Lemma 2. *Assume* $\mathbb{I} \models \mathsf{IsSetOfForm}(\mathbf{S})$. *The following are equivalent:*

1. $\mathbb{I} \models \mathsf{IsPartialValuation}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}})$.
2. *The mapping* $\mathbb{v}_{\mathbb{I}} : \Upsilon_{\mathbb{I}} \to A$ *defined by*

$$\mathbb{v}(\varphi_x^{\mathbf{S}}) := a \quad \textit{whenever } x_{\mathbb{I}} \in (\mathsf{V}_a)_{\mathbb{I}}$$

is well-defined and satisfies $\mathbb{v}_{\mathbb{I}} \in \mathsf{Val}_{\Upsilon_{\mathbb{I}}}(\mathbb{M})$.

Proof. For 1. implies 2., assume $\mathbb{I} \models \mathsf{IsSetOfForm}(\mathbf{S}) \wedge \mathsf{IsPartialValuation}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}})$. By line (11), every formula node is assigned a value. Well-definedness follows from line (12), ensuring each node has at most one truth-value, and from lines (13) and (16), guaranteeing that nodes corresponding to the same formula receive the same value. Furthermore, by lines (14), (17) and (18), $\mathbb{v}_{\mathbb{I}}$ respects the interpretation of connectives in $\mathbb{M}$, meeting the requirement in line (1). Finally, by lines (15) and (19), we conclude that $\mathbb{v}_{\mathbb{I}}(\Upsilon_{\mathbb{I}}) \in \mathcal{T}^*_{\mathbb{M}}$, ensuring $\mathbb{v}_{\mathbb{I}} \in \mathsf{Val}_{\Upsilon_{\mathbb{I}}}(\mathbb{M})$. For 2. implies 1., suppose $\mathbb{I} \not\models \mathsf{IsPartialValuation}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}})$. Then, at least one of the formulas in line (11)–(15) must fail in $\mathbb{I}$. If (11) does not hold, then some formula node lacks an assigned value, contradicting the well-definedness of $\mathbb{v}_{\mathbb{I}}$. If (12) or (13) fail, then $\mathbb{v}_{\mathbb{I}}$ assigns multiple values to the same formula and also contradicting that $\mathbb{v}_{\mathbb{I}}$ is well-defined. If (14) fails, then $\mathbb{v}_{\mathbb{I}}$ does not respect the interpretation connectives in $\mathbb{M}$, and if (15) fails then the values of the formulas in $\Upsilon_{\mathbb{I}}$ are not contained in a total component of $\mathbb{M}$, meaning in any case that $\mathbb{v}_{\mathbb{I}} \notin \mathsf{Val}_{\Upsilon_{\mathbb{I}}}(\mathbb{M})$. Hence, 2. cannot hold, completing the proof. □

Bivaluations We consider a new predicate B associated with partial bivaluations and define the following formulas

$$\mathsf{Bival}_{\mathbb{M}}(\mathbf{S}, \mathsf{B}) := \exists \mathbf{V}_{\mathbb{M}}.(\mathsf{IsPartialValuation}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}}) \wedge \mathsf{Fit}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}}, \mathsf{B})) \tag{20}$$

$$\mathsf{Fit}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}}, \mathsf{B}) := \forall x.(\mathsf{IsForm}(\mathbf{S}, x) \to \bigwedge_{a \in D} (\mathsf{V}_a(x) \to \mathsf{B}(x)) \wedge \bigwedge_{a \notin D} (\mathsf{V}_a(x) \to \neg \mathsf{B}(x))) \tag{21}$$

Lemma 3. *Assume that* $\mathbb{I} \models \mathsf{IsSetOfForm}(\mathbf{S})$. *The following are equivalent:*

1. $\mathbb{I} \models \mathsf{Bival}_{\mathbb{M}}(\mathbf{S}, \mathsf{B})$.
2. $\mathbb{b}_{\mathbb{I}} \in \mathsf{Bival}_{\Upsilon_{\mathbb{I}}}(\mathbb{M})$ *for* $\mathbb{b}_{\mathbb{I}} : \Upsilon_{\mathbb{I}} \to \{0, 1\}$ *defined by* $\mathbb{b}_{\mathbb{I}}(\varphi_x^{\mathbf{S}}) := 1$ *iff* $\mathsf{B}_{\mathbb{I}}(x)$.

Proof. From $\mathbb{I} \models \mathsf{IsSetOfForm}(\mathbf{S})$ if follows that $\Upsilon_{\mathbb{I}}$ is a finite set of formulas closed for subformulas. Further, for every $\mathbb{I}'$ coinciding with $\mathbb{I}$ on $\mathbf{S}$ we have $\Upsilon_{\mathbb{I}} = \Upsilon_{\mathbb{I}'}$. For 1. implies 2., assume that $\mathbb{I} \models \mathsf{Bival}_{\mathbb{M}}(\mathbf{S}, \mathsf{B})$. By line (20), there exists an interpretation $\mathbb{I}'$, differing from $\mathbb{I}$ only on predicates in $\mathbf{V}_{\mathbb{M}}$, such that

$$\mathbb{I}' \models \mathsf{IsPartialValuation}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}}) \wedge \mathsf{Fit}_{\mathbb{M}}(\mathbf{S}, \mathbf{V}_{\mathbb{M}}, \mathsf{B}).$$

By Lemma 2, we conclude that $\mathbb{v}_{\mathbb{I}'} \in \mathsf{Val}(\mathbb{M})_{\Upsilon_{\mathbb{I}}}$. Furthermore, by line (21), we have $\mathbb{b}_{\mathbb{I}}(\varphi) = 1$ iff $\mathbb{v}_{\mathbb{I}'}(\varphi) \in D$, for every $\varphi \in \Upsilon_{\mathbb{I}}$, and so $\mathbb{v}_{\mathbb{I}} = \mathbb{b}_{\mathbb{v}_{\mathbb{I}}}$. This establishes that $\mathbb{b}_{\mathbb{I}} \in \mathsf{Bival}_{\Upsilon_{\mathbb{I}}}(\mathbb{M})$.

For 2. implies 1., suppose $\mathbb{b} \in \mathsf{Bival}_{\mathcal{Y}_\mathbb{I}}(\mathbb{M})$. Then, by definition of $\mathsf{Bival}_{\mathcal{Y}_\mathbb{I}}(\mathbb{M})$, there exists a valuation $\mathbb{v} \in \mathsf{Val}(\mathbb{M})_{|\mathcal{Y}_\mathbb{I}}$, such that $\mathbb{b} = \mathbb{b}_\mathbb{v}$. Consider $\mathbb{I}'(\mathsf{V}_a) = \{\tau \in \mathsf{T}_\mathbb{I} : \mathbb{v}_\mathbb{I}(\varphi^\mathbb{I}_x) = a\}$, and $\mathbb{I}'(\mathsf{Q}) = \mathbb{I}(\mathsf{Q})$ for Q not in $\mathbf{V}_\mathbb{M}$. By Lemma 2, we have that $\mathbb{I}' \models \mathsf{IsPartialValuation}_\mathbb{M}(\mathbf{S}, \mathbf{V}_\mathbb{M})$. Further, using that $\mathbb{b} = \mathbb{b}_\mathbb{v}$, we have that $\mathbb{b}_\mathbb{I}(\varphi) = 1$ iff $\mathbb{v}_{\mathbb{I}'}(\varphi) \in D$ meaning that $\mathbb{I}' \models \mathsf{Fit}_\mathbb{M}(\mathbf{S}, \mathbf{V}_\mathbb{M}, \mathsf{B})$. Therefore, by line (20), we conclude that $\mathbb{I} \models \mathsf{Bival}_\mathbb{M}(\mathbf{S}, \mathsf{B})$. □

Comparing logics via WSkS

Proposition 2. *Given finite Σ-PNmatrices $\mathbb{M}_1$ and $\mathbb{M}_2$ let*

$$\Phi^n_{\mathbb{M}_1\mathbb{M}_2} := \forall \mathbf{S}\mathsf{B}.((\mathsf{IsSetOfForm}(\mathbf{S}) \wedge \mathsf{Bival}_{\mathbb{M}_2}(\mathbf{S}, \mathsf{B})) \rightarrow \mathsf{Bival}_{\mathbb{M}_1}(\mathbf{S}, \mathsf{B})).$$

Then, $\rhd^{|n}_{\mathbb{M}_1} \subseteq \rhd^{|n}_{\mathbb{M}_2}$ iff the formula $\Phi^n_{\mathbb{M}_1\mathbb{M}_2}$ is valid.

Proof. Since $\mathbb{M}_1$ and $\mathbb{M}_2$ are Σ-PNmatrices, we have that
For the left to right direction, assume that the formula $\Phi^n_{\mathbb{M}_1\mathbb{M}_2}$ is not valid, meaning that there is interpretation $\mathbb{I}$ such that

$$\mathbb{I} \models \mathsf{IsSetOfForm}(\mathbf{S}) \wedge \mathsf{Bival}_{\mathbb{M}_2}(\mathbf{S}, \mathsf{B}) \tag{22}$$

$$\mathbb{I} \not\models \mathsf{Bival}_{\mathbb{M}_1}(\mathbf{S}, \mathsf{B}) \tag{23}$$

Using (22) and Lemma 3 we have that $\mathbb{b}_\mathbb{I} \in \mathsf{Bival}_{\mathcal{Y}_\mathbb{I}}(\mathbb{M}_2)$. From (23) it follows that $\mathbb{I} \not\models \exists \mathbf{V}_{\mathbb{M}_1}.(\mathsf{IsPartialValuation}_{\mathbb{M}_1}(\mathbf{S}, \mathbf{V}_{\mathbb{M}_1}) \wedge \mathsf{Fit}_{\mathbb{M}_1}(\mathbf{S}, \mathbf{V}_{\mathbb{M}_1}, \mathsf{B}))$. Thus, by Lemma 3 we have that $\mathbb{b}_\mathbb{I} \notin \mathsf{Bival}_{\mathcal{Y}_\mathbb{I}}(\mathbb{M}_1)$. Hence, $\mathsf{Bival}_{\mathcal{Y}_\mathbb{I}}(\mathbb{M}_2) \not\subseteq \mathsf{Bival}_{\mathcal{Y}_\mathbb{I}}(\mathbb{M}_2)$ and by Lemma 1 we conclude that $\rhd^{|n}_{\mathbb{M}_1} \not\subseteq \rhd^{|n}_{\mathbb{M}_2}$.
For the other direction, assume that $\Phi^n_{\mathbb{M}_1\mathbb{M}_2}$ is valid and let $\mathbb{b} \in \mathsf{Bival}_\mathcal{Y}(\mathbb{M}_2)$ for some finite $\mathcal{Y} = \{\varphi_1, \ldots, \varphi_m\}$ with $m \in \mathbb{N}$. We consider the interpretation $\mathbb{I}$ such that $\mathsf{A}_\mathbb{I} = \{1^j : 0 \leq j \leq m-1\}$ (see Figs. 1 and 2), and set the interpretations of C occurring in $\mathbf{C}$ and P_i in $\mathbf{P}$, as necessary to append the tree representation of φ_j placing its root in the node $1^{j-1}2$ (see Figs. 2 and 3). Clearly, $\mathbb{I} \models \mathsf{IsSetOfForm}(\mathbf{S}) \wedge \mathsf{Bival}_{\mathbb{M}_2}(\mathbf{S}, \mathsf{B})$, $\mathcal{Y} = \mathcal{Y}_\mathbb{I}$ and $\mathbb{b} = \mathbb{b}_\mathbb{I}$. Since $\Phi^n_{\mathbb{M}_1\mathbb{M}_2}$ is valid we have that $\mathbb{I} \models \mathsf{Bival}_{\mathbb{M}_1}(\mathbf{S}, \mathsf{B})$. Hence, by Lemma 3 we obtain that $\mathbb{b} \in \mathsf{Bival}_\mathcal{Y}(\mathbb{M}_1)$, and so $\mathsf{Bival}_\mathcal{Y}(\mathbb{M}_2) \subseteq \mathsf{Bival}_\mathcal{Y}(\mathbb{M}_1)$.
Since this works for arbitrary finite $\mathcal{Y}$, we conclude that $\rhd^{|n}_{\mathbb{M}_1} \subseteq \rhd^{|n}_{\mathbb{M}_2}$ by Lemma 1.

5 Conclusion

We have shown that comparing the $\mathsf{Set} \times \mathsf{Set}$-logics characterized by finite PNmatrices is decidable when the language is restricted to a finite number of variables. However, the decidability of $\mathsf{Mult}^{|\omega}_{\subseteq}$, which drops the finite variables restriction, remains an open problem.

These results generalize what was known for *monadic* PNmatrices. In [7] it is shown that the $\mathsf{Set} \times \mathsf{Set}$-logics of finite monadic PNmatrices can be effectively finitely axiomatized, and only using $\mathsf{maxArity}(\Sigma)+1$ variables. Hence, to compare

the logics of finite monadic $\mathbb{M}_1$ and $\mathbb{M}_2$ we just have to generate the axiomatization of $\mathbb{M}_1$ and check whether every generated rule is sound in $\mathbb{M}_2$. The present results show that we can alternatively rely on an algorithm deciding the validity of $\Phi^n_{\mathbb{M}_1\mathbb{M}_2}$ for $n = \mathsf{maxArity}(\Sigma) + 1$.
In conclusion, to determine if $\rhd_{\mathbb{M}_1} \subseteq \rhd_{\mathbb{M}_2}$ we do not really need to know a specific axiomatization of $\rhd_{\mathbb{M}_1}$ (which may not even be finitely $\mathsf{Set} \times \mathsf{Set}$-axiomatizable) but only require a finite bound on the necessary number of variables. This naturally raises the question of whether every finite PNmatrix $\mathbb{M}$ is axiomatizable using a finite number of variables that can be computed from $\mathbb{M}$. In light of Theorem 1, if $\mathsf{Mult}^{\omega}_{\subseteq}$ turns out to be undecidable, then the answer to this question must be negative.

Acknowledgments

This work is funded by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., and, when eligible, co-funded by EU funds under project/support UID/50008/2025 – Instituto de Telecomunicações

References

1. Avron, A.: The normal and self-extensional extension of Dunn–Belnap logic. Logica Universalis **14**, 281–296 (2020)
2. Avron, A., Arieli, O., Zamansky, A.: Theory of Effective Propositional Paraconsistent Logics, Studies in Logic, vol. 75. College Publications, London (2018)
3. Avron, A., Lev, I.: Non-deterministic multiple-valued structures. Journal of Logic and Computation **15**(3), 241–261 (2005)
4. Avron, A., Zohar, Y.: Rexpansions of non-deterministic matrices and their applications in non-classical logics. Review of Symbolic Logic **12**(1), 173–200 (2019)
5. Baaz, M., Lahav, O., Zamansky, A.: Finite-valued semantics for canonical labelled calculi. Journal of Automated Reasoning **51**(4), 401–430 (2013)
6. Caleiro, C., Filipe, P., Marcelino, S.: Equivalence of finite non-deterministic logical matrices is undecidable. Studia Logica (2025), `https://arxiv.org/abs/2412.14057`, in print
7. Caleiro, C., Marcelino, S.: Analytic calculi for monadic PNmatrices. In: Logic, Language, Information, and Computation (WoLLIC 2019). Lecture Notes in Computer Science, vol. 11541, pp. 84–98. Springer (2019)
8. Caleiro, C., Marcelino, S.: On axioms and rexpansions. In: Arieli, O., Zamansky, A. (eds.) Arnon Avron on Semantics and Proof Theory of Non-Classical Logics, Outstanding Contributions to Logic, vol. 21. Springer (2021)
9. Caleiro, C., Marcelino, S.: Modular many-valued semantics for combined logics. Journal of Symbolic Logic **89**(2), 583–636 (2024)
10. Caleiro, C., Marcelino, S., Marcos, J.: Combining fragments of classical logic: When are interaction principles needed? Soft Computing **23**(7), 2213–2231 (2019)
11. Caleiro, C., Marcelino, S., Rivieccio, U.: Some more theorems on structural entailment relations and non-deterministic semantics. In: Malinowski, J., Palczewski, R. (eds.) Janusz Czelakowski on Logical Consequence, Outstanding Contributions to Logic, vol. 27, pp. 345–375. Springer (2011)

12. Ciabattoni, A., Lahav, O., Spendier, L., Zamansky, A.: Taming paraconsistent (and other) logics: An algorithmic approach. ACM Transactions on Computational Logic **16**(1), 5:1–5:23 (2014)
13. Coniglio, M.E., del Cerro, L.F., Peron, N.M.: Finite non-deterministic semantics for some modal systems. Journal of Applied Non-Classical Logics **25**(1), 20–45 (2015)
14. Coniglio, M.E., Toledo, G.V.: Two decision procedures for da Costa's C_n logics based on restricted Nmatrix semantics. Studia Logica **110**(3), 601–642 (2022)
15. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research, Lecture Notes in Computer Science, vol. 2500. Springer (2002)
16. Grätz, L.: Truth tables for modal logics T and S4, by using three-valued non-deterministic level semantics. Journal of Logic and Computation **32**(1), 129–157 (Jan 2022)
17. Greati, V., Marcelino, S., Marcos, J., Rivieccio, U.: Adding an Implication to Logics of Perfect Paradefinite Algebras. Mathematical Structures in Computer Science **34**(10), 1138–1183 (2024)
18. Humberstone, L.: The Connectives. MIT Press (2011)
19. Ivlev, J.: A semantics for modal calculi. Bulletin of the Section of Logic **17**(3–4), 114–121 (1988)
20. Kearns, J.: Modal semantics without possible worlds. Journal of Symbolic Logic **46**(1), 77–86 (1981)
21. Lahav, O., Avron, A.: A unified semantic framework for fully structural propositional sequent systems. ACM Transactions on Computational Logic **14**(4), 271–273 (2013)
22. Lahav, O., Zohar, Y.: Effective semantics for the modal logics K and KT via non-deterministic matrices. In: Blanchette, J., Kovács, L., Pattinson, D. (eds.) Automated Reasoning. pp. 468–485. Springer International Publishing, Cham (2022)
23. Marcelino, S.: An unexpected Boolean connective. Logica Universalis **16**, 85–103 (2022)
24. Marcelino, S., Caleiro, C.: Disjoint fibring of non-deterministic matrices. In: Kennedy, J., de Queiroz, R. (eds.) Logic, Language, Information, and Computation (WoLLIC 2017). Lecture Notes in Computer Science, vol. 10388, pp. 242–255. Springer (2017)
25. Marcelino, S., Caleiro, C.: Axiomatizing non-deterministic many-valued generalized consequence relations. Synthese **198**(22), 5373–5390 (2021)
26. Marcelino, S., Caleiro, C., Filipe, P.: Computational properties of finite PNmatrices. Journal of Logic and Computation **23**(8), 1694–1719 (2022)
27. Omori, H., Skurt, D.: More modal semantics without possible worlds. IfCoLog Journal of Logics and their Applications **3**(5), 815–846 (2016)
28. Pawlowski, P., Skurt, D.: 8-valued non-deterministic semantics for modal logics. Journal of Philosophical Logic **53**, 351–371 (2024)
29. Rabin, M.O.: Decidability of Second-Order Theories and Automata on Infinite Trees. Transactions of the American Mathematical Society **141**, 1–35 (1969)
30. Scott, D.: Completeness and axiomatizability in many-valued logic. In: Proceedings of the Tarski Symposium, Proceedings of Symposia in Pure Mathematics, vol. 25, pp. 411–435. American Mathematical Society, Berkeley, CA (1974)
31. Shoesmith, D., Smiley, T.: Multiple-Conclusion Logic. Cambridge University Press, Cambridge (1978)
32. Sipser, M.: Introduction to the Theory of Computation. Cengage Learning, 3 edn. (2013)

33. Thomas, W.: Automata on Infinite Objects. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, pp. 133–191. Elsevier (1990)
34. Wroński, A.: A three element matrix whose consequence operation is not finitely based. Bulletin of the Section of Logic **8**(2), 68–70 (1979)
35. Zohar, Y.: Gentzen-type proof systems for non-classical logics. PhD Thesis, Tel Aviv University, Tel Aviv, Israel (2018)

goLoop: SMT-Based Loop Analysis via Global Optimization

Markus Krahl[1,2] [0009−0009−2435−1108], Matthias Güdemann[1] [0000−0002−1002−6023], and Stefan Wallentowitz[1] [0000−0003−3182−4929]

[1] Hochschule München University of Applied Sciences, Munich, Germany
{markus.krahl,matthias.guedemann,stefan.wallentowitz}@hm.edu
[2] TASKING Germany GmbH, Munich, Germany
markus.krahl@tasking.com

Abstract. Reconfigurable transition systems (RTS) model dynamic behaviors where state transitions and system configurations evolve during execution. Formal reasoning about such systems often requires analyzing loops with a varying number of iterations. Although these loops terminate, they lack a fixed bound and are therefore difficult to analyze using bounded model checking (BMC) techniques, which rely on loop unrolling, thereby facing scalability limitations.
We propose a global optimization (GO) based model checking approach designed to efficiently handle these loop-constructs without explicit unrolling. Loops are treated as executable constructs and, together with verification conditions, are encoded as optimization problems We formalize the approach with an extended SMT syntax supporting loop constructs, define rules for translating quantifier-free SMT equations involving floating point (FP) and bitvector theories into GO functions, and demonstrate its effectiveness on loops performing FP computations.

1 Introduction

Modern software and cyber-physical systems are increasingly required to adapt their behavior dynamically in response to internal conditions or external events. This adaptability is often modeled using reconfigurable transition systems (RTS), where the set of states, their properties, or the transition relations may change during execution. Unlike traditional static transition systems, RTS capture structural modifications that occur as the system evolves. Such reconfigurations are regularly preceded by complex computations that must finish before the system can safely change its configuration. For instance, cyber-physical systems may perform self-healing strategies before activating a different control strategy [12]. To ensure correctness of these systems, formal verification methods must be able to analyze these pre-reconfiguration computations, which could involve long, terminating but varying bounded loops.

Bounded Model Checking (BMC) is a widely used approach for verifying safety properties of finite-state systems [3]. It systematically explores program executions up to a fixed bound by unrolling loops and encoding the resulting

verification conditions (VC) into satisfiability (SAT) or Satisfiability Modulo Theories (SMT) problems, enabling reasoning over richer domains such as floating point (FP), integers, arrays, and bitvectors (BV). If a property is violated, BMC produces a counterexample, illustrating a sequence of states that leads to the violation. While BMC has proven highly effective for finding counterexamples within bounded execution paths, it is incomplete due to the under-approximation of the search space; the absence of a counterexample up to a certain bound does not guarantee the correctness of the system for larger bounds. In systems that undergo reconfiguration, loop intensive computations often precede behavior leading to structural changes. Capturing such behavior with BMC requires large unrolling depths, which exponentially increase the size of the verification formula and make analysis infeasible due to its poor scalability with respect to increasing bounds.

In this paper, we present a novel, alternative approach called *goLoop* that formulates the verification task as a global optimization (GO) problem, with the goal of analyzing loop constructs without the need for explicit unrolling or induction-based reasoning, thereby addressing key limitations of existing techniques. Instead of transforming loops into static unrolled structures potential leading to exponential blow-up in the size of verification conditions, we propose to execute them within a GO framework extending the key concepts presented in XSat [7], goSAT [2], and further developed in parSAT [13]. This enables running the optimization function in compiled form that would allow to use native instructions of the platform resulting in faster solving speed; for instance using FP instructions of the CPU instead of modelling their behavior with BVs if the to be analyzed loop or program contains FP computations. Additionally, we suggest additions to the SMT-LIB2 standard to enable a more imperative style for syntactically supporting loop expressions. *goLoop* shall serve as building block for future verification techniques that combine optimization-based reasoning with BMC techniques.

The paper is structured as follows. Section 2 discusses related work. In section 3 we contend that existing BMC techniques may exhibit limitations in verifying algorithms involving long running loops, particularly performing FP computations. The theoretical foundation and core principles of our GO approach for evaluating loop-based programs and corresponding exemplary applications are presented in section 4.

2 Related Work

Previous research has explored loop summarization and acceleration methods, where the original loop construct is replaced by a model such as [15], [10], or [6]. Additionally, [14] suggests adding auxiliary paths to loops for under-approximation that could be used complementary with other verification methods. Combining k-induction with loop invariant generation [1] has shown potential to enable BMC to verify safety properties in programs involving loops without manual annotations. [17] presents a BMC approach utilizing interpolation-based

summaries to analyze program behavior involving loops. In [20] path dependency analysis is used to perform loop summarization which shows promising results in verification of loop-based programs. Recent approaches, such as [19], explore loop invariant generation by Large Language Models (LLM), bridging the gap between automated reasoning and machine learning. While all of these approaches have demonstrated promising results, their practical application and common usage in industry has remained limited.

The idea of formulating FP constraints as GO problems was first introduced in XSat [7]. It solves SMT equations in pure FP theory by transforming a quantifier-free formula $F(\overrightarrow{x})$, where $\overrightarrow{x} \in FP^n$, into an equisatisfiable objective function $G(\overrightarrow{x})$. $G(\overrightarrow{x})$ is minimized using GO; a global minimum of zero indicates that $F(\overrightarrow{x})$, is satisfiable, while a non-zero minimum suggests unsatisfiability. XSat takes SMT-LIB2 input and generates C code for the objective function, which is compiled and executed as a Python module using the `basinhopping` from SciPy [18]. goSAT [2] builds on XSat, applying the same transformation for SMT equations in FP theory. It also supports code generation but introduces Just-in-Time (JIT) compilation of the objective function for immediate evaluation. In contrast to XSat, goSAT allows selection among multiple GO algorithms via the NLopt library [11]. parSAT [13] is an integrated tool that performs a portfolio-based semi-decision procedure for SMT equations in FP theory, extending the concepts presented in XSat and goSAT. It employs multiple instances of GO methods in a concurrent setting to solve a given SMT equation translated into a GO problem.

Our work extends the approach presented in XSat and goSAT by focusing on the representation and execution of varying bounded but terminating loops directly within a GO framework. Besides, we present extensions to the SMT-LIB2 standard to improve its modelling capabilities for loop constructs within the analyzed program. By doing so, we aim to provide a complementary approach to current BMC techniques.

3 Motivation and Background

In this work we consider only safety properties (invariants). While this restricts the expressiveness, in practical applications this is often enough to express the desired properties. This problem can be expressed in terms of reachability and allows for using satisfiability-based reasoning, allowing for much larger state-spaces to be analyzed in comparison to techniques that require construction of the entire state space, e.g., for BDD-based reasoning.

3.1 Bounded Model Checking

In BMC [3, 4], the analyzed program is modelled as a state transition system: A state transition system is defined as $M = (S, T, I)$ where S is the set of states, $I \subseteq S$ is the set of initial states, and $T \subseteq S \times S$ is the transition relation. Given a transition system M, a property ϕ, and a bound k, BMC unrolls the system

k times and generates a VC ψ that is satisfiable if and only if there exists a counterexample of ϕ for a bound lower than or equal to k . ψ is a quantifier-free formula in a decidable subset of first-order logic, which is then checked for satisfiability by an SMT solver. This results in a model checking problem denoted by the following logical formula:

$$\psi = I(s_0) \wedge \bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg\phi(s_i)$$

This formula would be `SAT` if there exists a possible path $\pi = (s_0, s_1..., s_k)$ from I to $\neg\phi$ in the program, such that the property ϕ does not hold. Each satisfiable assignment would represent a valid counterexample. In case this equation is `UNSAT`, the property ϕ holds for the analyzed program unrolled by k.

The following example – while involving relatively simple code – demonstrates some difficulties that might occur when analyzing loop-based programs with BMC. We use the C Bounded Model Checker (CBMC) [5] tool together with an SMT solver as backend to illustrate the behavior of BMC. The C implementation in Listing 1 calculates the harmonic series $\sum_{k=1}^{n} = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$. It contains an assertion that its result is always smaller than 4.0. However, the result of the harmonic series diverges to ∞ as n tends to ∞. Therefore, the given assertion in Listing 1 should not hold for values of $x > 30$.

Listing 1 C implementation of harmonic series with assertion

```
unsigned int x = nondet_uint();
float y = 0.0f;

while (x >= 1)
{
  y += (1.0f / x );
  x -= 1;
}

assert(y < 4.0f);
```

Using CBMC to verify this assertion would produce the SMT equation shown in Listing 2, which encodes the harmonic series constraints in a syntax similar to SMT-LIB2 (slightly simplified for readability – for more details on the encoding see [9]). With rounding set to `nearest even` (RNE), the while-loop is unrolled three times (bound $k = 3$). Each iteration introduces a new boolean guard variable, modeling loop behavior up to the bound. When submitted to the solver, the formula is `UNSAT` because the guards implicitly restrict x to a small range $[0, 2]$, where the asserted property holds. Solving required approximately 0.1 seconds on our system.

Listing 2 SMT formula for assertion of harmonic series

```
(declare-fun |guard#1| () Bool)
(declare-fun |guard#2| () Bool)
(declare-fun |guard#3| () Bool)
(define-fun B11 () Bool (and |guard#1| |guard#2| |guard#3|))
(assert (not B11)) ; for bounding the loop at least one guard must be false
(declare-fun |x1| () (_ BitVec 32)) ; input variable x1 as bitvector
(assert (= |guard#1| (>= |x1| 1 ))) ; first guard for first loop iteration
(define-fun |y1| () (_ FloatingPoint32) (+ 0.0 (/ 1.0 (to_fp_32 |x1|))))
(define-fun |x2| () (_ BitVec 32) (- |x1| 1))
(assert (= |guard#2| (>= |x2| 1))) ; second guard for second loop iteration
(define-fun |y2| () (_ FloatingPoint32) (+ |y1| (/ 1.0 (to_fp_32 |x2|))))
(define-fun |x3| () (_ BitVec 32) (- |x2| 1))
(assert (= |guard#3| (>= |x3| 1))) ; loop is finally bounded by third guard
(define-fun |y3| () (_ FloatingPoint32) (ite (and |guard#1| (not |guard#2|)) |y1| |y2|))
(define-fun |y4| () (_ FloatingPoint32) (ite |guard#1| |y3| 0.0))
(assert (not (< |y4| 4.0)))
(check-sat)
```

Increasing the input domain requires larger unrolling depths. For $x \in [0, 20]$, k must be 21 for which solving required 2.4 seconds and gave `UNSAT`. To detect a counterexample for $x \in [0, 31]$, k must be incremented to 32, leading to a 5.7 second runtime. In this case, the solver reported a counterexample for $x = 31$, yielding $y \approx 4.027$, which violates the assertion.

While BMC is effective for programs without loops or with easily determined bounds, analyzing loop-intensive code is challenging. Large bounds may be needed to expose violations occurring deep in the execution, resulting in significant formula growth, longer solving times, or out-of-memory errors. Conversely, small bounds risk missing property violations entirely. Choosing different bounds for different loops quickly gets very challenging.

3.2 goLoop Approach

To tackle this problem we propose formulating the model checking problem as a global constraint optimization problem and to extend existing bounding checking techniques with the capability to analyze loops directly without unrolling.

The verifier seeks a satisfiable variable assignment of the VC's formula by minimizing an objective function to represent the violation of the VC. Loops are not unrolled but included as sub-functions in the objective functions, such that they are executed under constraints by the GO framework. Each loop encoded as sub-function represents a part of the objective function. Accordingly, based on the control flow graph (CFG) – including loops – of the analyzed program and the property ϕ, the objective function $F(x)$ is constructed for the VC ψ, where the vector x denotes the program's input. $F(x)$ quantifies the distance dependent on the program input x from violating the property ϕ. When $F(z) = 0$, i.e., the optimization function has a global minimum of 0 for the program input z, then z presents a counterexample that would lead to a violation of property ϕ. This results in the following formula to express the VC ψ:

$$\psi := \min_{x \in R} F(x) = 0$$

where the set R represents the possible input domain for the program input x.

Accordingly, the harmonic series implementation shown in Listing 1 can first be transformed into an SMT equation with syntactic support to include loops (denoted by *SMT-Loop*), as illustrated in Listing 3. Further details on the adaptations to the SMT-LIB2 syntax are provided in Section 4.

Listing 3 SMT-Loop program for assertion of harmonic series with inclusion of loop

```
(declare-fun |x_in| () (_ BitVec 32)) ; input variable x_in as bitvector
(define-fun-mut |x| () (_ BitVec 32) |x_in|) ; convert to mutable
(define-fun-mut |y| () (_ FloatingPoint32) (0.0))
(define-fun-mut |loop_cond| () Bool (>= |x| 1))
(loop (|loop_cond|)
  (define-fun-mut |y| () (_ FloatingPoint32) (+ |y| (/ 1.0 (fp_32 |x|))))
  (define-fun-mut |x| () (_ BitVec 32) (- |x| 1))
  (define-fun-mut |loop_cond| () Bool (>= |x| 1))
)
(assert (not (< |y| 4.0)))
(check-sat)
```

Second, following our idea of *goLoop*, the SMT equation can be converted into an objective function $F(x)$ whose plot over the interval $x \in [0, 39]$ is presented in Figure 1. The transformation from an SMT equation with loop constructs into an optimization function is described in detail in Section 4. As depicted in the plot, the function reaches a value of zero for all $x \geq 31$, with $F(31)$ marked by a red dot. That indicates that every $x \in [31, \infty)$ corresponds to a satisfying assignment of the SMT equation from Listing 3 and, thus, represents a counterexample to the assertion $y < 4.0$. GO algorithms can therefore be used to efficiently locate such zero-valued global minima. For example, the `basinhopping` algorithm from the SciPy Python package required less than 0.025 seconds on our system to find a global minimum where $F(x) = 0$.

4 Theoretical Considerations

In the following, we present our considerations for integrating loop semantics into SMT equations and to construct the corresponding objective functions.

4.1 Transforming an SMT Equation into a GO Problem

In general, a quantifier-free SMT equation $F(\overrightarrow{x})$ with $\overrightarrow{x} \in P^n$, where P^n represents the set of the program input domain, is transformed into a mathematical objective function $G(\overrightarrow{x})$. Computing $G(\overrightarrow{x})$ with a given input vector $\overrightarrow{a}$ either

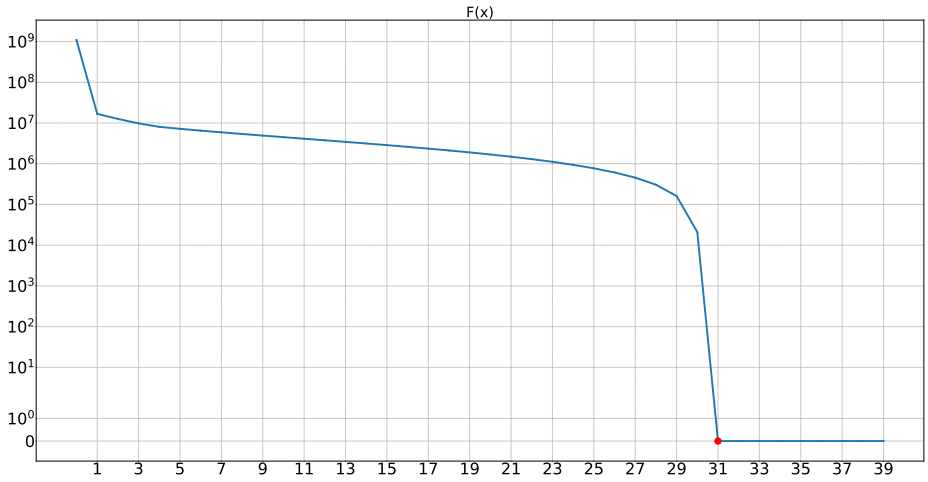


Fig. 1: Plot of F(x) for the SMT equation in Listing 3

returns 0 if $\overrightarrow{a}$ corresponds to a *satisfiable* assignment α for $F(\overrightarrow{x})$ or a positive distance value. To ensure equivalence between the optimization function $G(\overrightarrow{x})$ and the initial SMT formula $F(\overrightarrow{x})$, the following requirements, which are derived from those originally specified in XSat, must be satisfied: the objective function must always return a non-negative value, and it must return zero if and only if the corresponding valuation of the free variables is a satisfying assignment of $F(\overrightarrow{x})$.

The language $\mathcal{L}_{GO-SMT}$ defines a simplified SMT syntax based on the SMT-LIB2 standard. It generalizes a quantifier-free SMT formula $F(\overrightarrow{x})$ involving FP and BV theories. $\mathcal{L}_{GO-SMT}$ extends the language supported by goSAT and XSat, its syntax is defined as:

$$
\begin{array}{ll}
\textbf{\textit{Bool}} & \pi := \neg\pi \mid \pi_1 \odot \pi_2 \mid f_1 \bowtie_{FP} f_2 \mid e_1 \bowtie_{BV} e_2 \mid T \mid F \mid fun_{Bool}((e_n)^*, (f_n)^*, (\pi_n)^*) \\
\textbf{\textit{FP}} & f := c_f \mid v_f \mid f_1 \otimes_{FP} f_2 \mid fun_{FP}((f_n)^*, (e_n)^*, (\pi_n)^*) \mid ite(\pi, f_1, f_2) \\
\textbf{\textit{BV}} & e := c_e \mid v_e \mid e_1 \otimes_{BV} e_2 \mid fun_{BV}((e_n)^*, (f_n)^*, (\pi_n)^*) \mid ite(\pi, e_1, e_2)
\end{array}
$$

where $\odot \in \{\wedge, \vee\}$, $\bowtie_{FP} \in \{<, \leq, >, \geq, ==, \neq\}$ (for FP types), $\bowtie_{BV} \in \{<_{\{u|s\}}, \leq_{\{u|s\}}, >_{\{u|s\}}, \geq_{\{u|s\}}, ==, \neq\}$ (the subscript $\{u|s\}$ defines the respective operand for signed and unsigned BV types), $\otimes_{FP} \in \{+, -, *, /\}$, and $\otimes_{BV} \in \{+, -, *, /_{\{u|s\}}, <<, >>_{\{u|s\}}, rem_{\{u|s\}}, \&, |, \sim\}$. c_f represents a FP constant, c_e a BV constant, v_f is a FP variable, and v_e a BV variable. *ite* corresponds to the *if-then-else*-function defined in the SMT-LIB2 core theory. It returns the first expression (e_1 or f_1, for either BV or FP type) if the Boolean argument π is true; otherwise, it returns the second expression (e_2 or f_2, depending on the type). fun_{Bool}, fun_{FP}, or fun_{BV} represent interpreted functions. They support an arbitrary number of different typed arguments (arguments of BV type represented by e_n, FP type f_n, and boolean type π_n) and may be used to represent further

operations as stated in the SMT-LIB2 standard, i.e., `fp.roundToIntegral` or `fp.isNaN` from the FP theory. Furthermore, a user may also provide individual specified functions, which are currently not part of the SMT-LIB2, such as the natural logarithm function `ln`.

Based on $\mathcal{L}_{GO-SMT}$, we define $F(\overrightarrow{x})$ as conjunction of the asserted boolean constraints of an SMT equation from where we deduce the optimization function $G(\overrightarrow{x})$ as follows:

$$F(\overrightarrow{x}) = \bigwedge_{i \in I} \pi_i \quad \rightarrow \quad G(\overrightarrow{x}) = \sum_{i \in I} d(\pi_i)$$

where the distance function $d(\pi_i)$ translates the expression denoted by the boolean symbol π_i to a real value that is equal to or greater than zero. A value of zero for $d(\pi_i)$ implies that expression of the symbol π_i is `true`, whereas $d(\pi_i) > 0$ states that the expression denoted by π_i is `false`. Therefore, if $G(\overrightarrow{x}) == 0$, it can be deduced that all boolean constraints of $F(\overrightarrow{x})$ are `true`. Otherwise, if $G(\overrightarrow{x}) > 0$, some of the boolean constraints in the conjunction of $F(\overrightarrow{x})$ are `false`. First, we define the auxiliary functions $\theta_{\{U|S\}BV}(e_1, e_2)$ that compute the distance value between the either signed or unsigned BVs e_1 and e_2.

$$\theta_{UBV}(e_1, e_2) = \begin{cases} \texttt{toUInt}(e_1) - \texttt{toUInt}(e_2) \text{ if } & \texttt{toUInt}(e_1) > \texttt{toUInt}(e_2) \\ \texttt{toUInt}(e_2) - \texttt{toUInt}(e_1) \text{ else} \end{cases}$$

$$\theta_{SBV}(e_1, e_2) = \begin{cases} \texttt{toInt}(e_1) - \texttt{toInt}(e_2) \text{ if } & \texttt{toInt}(e_1) > \texttt{toInt}(e_2) \\ \texttt{toInt}(e_2) - \texttt{toInt}(e_1) \text{ else} \end{cases}$$

where the function `toUInt` transforms a given unsigned BV to a number $n \in \mathbb{N}_0$ and `toInt` calculates a number $z \in \mathbb{Z}$ for a signed BV. It shall be noted that both signed and unsigned variants of θ always return non-negative values, evaluate to zero if and only if their arguments are equal, and produce identical results regardless the order of their arguments.

We define a distance function $d(\pi)$ mapping boolean constraints to non-negative real values, ensuring that $d(\pi) = 0$ iff the boolean constraint π is true. For compound boolean expressions, we use additive and multiplicative compositions (e.g., $d(\pi_1 \wedge \pi_2) = d(\pi_1) + d(\pi_2)$, $d(\pi_1 \vee \pi_2) = d(\pi_1) * d(\pi_2)$). Negation is handled by inversion $d(\neg\pi) = 0$ if $d(\pi) > 0$, and 1 otherwise. For BVs, $d(\_)$ is defined analogously utilizing the previously type-aware distance function $\theta_{\{U|S\}BV}(\_, \_)$, that measures the difference between unsigned or signed operands for a given operator, such as for the operand $<$ and $\leq$:

$$d(e_1 <_{\{u|s\}} e_2) = \begin{cases} 0 & \text{if } \quad e_1 <_{\{u|s\}} e_2 \\ \theta_{\{U|S\}BV}(e_1, e_2) + 1 & \text{else} \end{cases}$$

$$d(e_1 \leq_{\{u|s\}} e_2) = \begin{cases} 0 & \text{if } \quad e_1 \leq_{\{u|s\}} e_2 \\ \theta_{\{U|S\}BV}(e_1, e_2) & \text{else} \end{cases}$$

Similar definitions apply for $>, \geq, ==, \neq$ and are reused for FP numbers after converting the FP operands to unsigned BVs. Due to the special behavior of non-finite FP numbers, such as NaN or $\pm\infty$, these distance functions are only valid

for finite FP values. Considering all defined variants of $d(\_)$ and $\theta_{\{U|S\}BV}(\_,\_)$, we can conclude that they satisfy the principles required to ensure equivalence between the optimization function $G(\overrightarrow{x})$ and the initial SMT formula $F(\overrightarrow{x})$.

4.2 Integration of Loop Semantics

We define $\mathcal{L}_{Loop-SMT}$ as an extension of $\mathcal{L}_{GO-SMT}$ by adding a boolean *loop* statement such as,

$$\textbf{\textit{Bool}} \quad \pi := \mathit{loop}(\pi_1, ((e_n)^*, (f_n)^*, (\pi_n)^*))$$

which has a Boolean type and becomes `true` when the loop terminates, i.e., the loop condition becomes `false`. It requires a Boolean condition π_1 and repeats the additionally provided arbitrary number of statements until π_1 becomes `false`. The *loop* command may redefine the symbols of expressions used as arguments implicitly during the iterations of the loop. In other words, it may represent a function of Boolean type that modifies the symbols given as parameters. By doing so, we can model the side effects the loop imposes on the whole formula in the optimization function. For instance the following *loop*-statement $\mathit{loop}((i < 10), ((i \leftarrow i + 1)))$ would increment the variable i ten times by 1 and return `true` after the loop has terminated such that $i := 10$.

The loop-statement is treated as interpreted function that repeatedly executes the statements in $((e_n)^*, (f_n)^*, (\pi_n)^*)$ and returns true iff the corresponding loop terminates. When the loop terminates, its iteration condition π_1 must be `false` afterwards. Therefore, $d(\mathit{loop}(\pi_1, ((e_n)^*, (f_n)^*, (\pi_n)^*)))$ can be expressed by $d(\neg\pi_1)$ which returns 0 if and only if the interpreted loop statement terminated.

To simplify transforming loop-based programs into SMT equations, we propose to extend SMT-LIB2 with an experimental layer providing imperative semantics (at the moment incompatible with the underlying logic of available SMT solvers). The current standard requires each variable to have a unique symbol name, which becomes inefficient when variables are updated multiple times within loops. In traditional bounded model checking, this is handled by unrolling loops and introducing new symbols for each iteration. However, our approach executes loops directly, allowing an unknown number of loop iterations which makes static symbol creation impractical. We therefore introduce a new mutable definition statement: `(define-fun-mut symbol-name)` which, unlike `(define-fun symbol-name)`, allows variable reassignment. Additionally, we propose a loop construct as extension to SMT-LIB2: `(loop loop-cond)(loop-iteration-statement`$^+$`)`,
where `loop-cond` is the loop condition variable and `loop-iteration-statement`$^+$ represents one or many statements which are evaluated in each loop iteration. The `loop`-command returns `true` when the loop terminates. Consequently, it should not be used as the loop-condition variable within another `loop`-command.

We illustrate these considerations with an example involving a C implementation (Listing 4) that tracks elapsed time in seconds using FP arithmetic, inspired by the Patriot missile system bug [8]. The program contains a loop that "measures" elapsed time in increments of 0.1 seconds up to a user-defined bound `n_bound`.

To verify correctness, the program asserts that the absolute error between the measured and expected time remains within a tolerance of 10^-2 seconds. However, due to the behavior of FP arithmetic, the assertion may fail for large values of `n_bound`. In particular, the value 0.1 cannot be represented exactly in FP format, leading to the accumulation of rounding errors across iterations. Moreover, repeated additions of this small constant to the growing FP variable `time_seconds` eventually cause value cancellation, such that `time_seconds` remains the same value after the addition. Determining the number of loop iterations (i.e., the value of `n_bound`) required to violate the assertion is challenging for BMC tools, as it requires a large unrolling depth k.

Listing 5 presents the corresponding SMT formula, which includes the original, unmodified loop when applying the *goLoop* approach (denoted by *SMT-Loop*). Following the transformation rules for constructing an objective function from an SMT equation, Figure 2 depicts the resulting function $F(x)$ for values of $x \in [0, 3500]$, where x represents `n_bound`. The red dot marks the first global minimum of zero at $x = 3149$. With an appropriate GO algorithm, such zero-valued minima can be located, corresponding to satisfying assignments of the SMT formula in Listing 5 and thus counterexamples to the assertion in Listing 4. Despite the presence of multiple local minima in $F(x)$, the `basinhopping` GO method required less than 1.2 seconds on our system to locate a zero-valued global minimum. In contrast, CBMC with a bound of $k = 3150$ required approximately 1.4 minutes to find the first counterexample.

Listing 4 C implementation of counting time by 0.1 second increments

```
unsigned int n_bound = nondet_uint();
unsigned int time_ticks = 0;
float time_seconds = 0.0f;
double tol = 1e-2;

while (time_ticks < n_bound)
{
  time_seconds += 0.1f;
  time_ticks++;
}

double expected = ((double) time_ticks / 10.0);
double err = fabs((double) time_seconds - expected);
assert(err <= tol);
```

5 Strengths and Limitations

Overall, the proposed *goLoop* approach demonstrates promising potential. Due to the specified translation rules for converting an SMT equation in FP or BV theory into an optimization function, no further adaptations or modifications of the involved loop constructs in the analyzed program are required. Furthermore,

Listing 5 SMT-Loop program for assertion in counting time code

```
(declare-fun |n_bound| () (_ BitVec 32)) ; input variable n_bound as bitvector
(define-fun-mut |time_ticks| () (_ BitVec 32) 0)
(define-fun-mut |time_seconds| () (_ FloatingPoint32) (0.0))
(define-fun |tol| () (_ FloatingPoint64) (1e-2))
(define-fun-mut |loop_cond| () Bool (< |time_ticks| |n_bound|))
(loop (|loop_cond|)
  (define-fun-mut |time_seconds| () (_ FloatingPoint32) (+ |time_seconds| 0.1))
  (define-fun-mut |time_ticks| () (_ BitVec 32) (+ |x| 1))
  (define-fun-mut |loop_cond| () Bool (< |time_ticks| |n_bound|))
)
(define-fun |expected| () (_ FloatingPoint64) ((fp_64 (/ |time_ticks| 10.0))))
(define-fun |err| () (_ FloatingPoint64) (fp.abs(- (fp_64 time_seconds) |expected|)))
(assert (not (<= |err| |tol|)))
(check-sat)
```

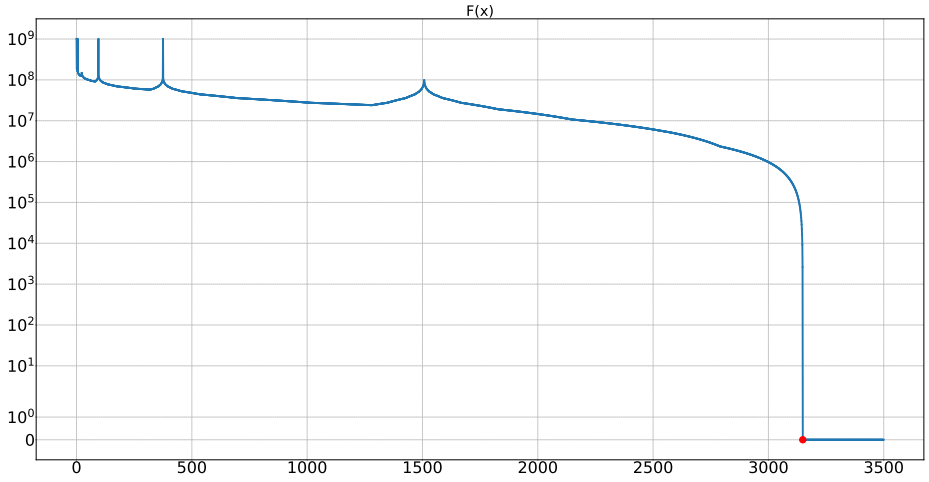


Fig. 2: Plot of F(x) for the SMT equation in Listing 5

any derivative-free GO algorithm could be used to systematically search for the global minimum of the generated optimization function. *goLoop* also supports compiling the generated optimization function into a native binary, improving execution speed by leveraging the instruction set of the target verification platform – for example, executing FP instructions directly rather than converting them into satisfiability problems, as some SMT solvers do via word- or bit-blasting (e.g. [16]). This further enables the support for dependency functions – whether used in the loop body or elsewhere in the analyzed program – that exist only in compiled form or are provided by specific hardware devices, such as functions from mathematical libraries. Moreover, *goLoop* offers the potential for parallel optimization of the same optimization function by leveraging different GO methods. This also simplifies the exchange of already evaluated information between the various GO instances, compared to the more complex information exchange between different SMT solvers.

Nevertheless, *goLoop* has certain limitations. Programs with infinite loops cannot be analyzed, as the generated optimization function would still include the unbounded loop, preventing the GO process from terminating. Likewise, using *goLoop* to verify loop termination is limited; setting a timeout for the optimization function is insufficient without additional knowledge of the program and its runtime behavior. Finding a loop variant to prove termination is generally undecidable. Another limitation, is the undecidability of the underlying problem. Due to the many conditionals in the equations used to construct the optimization function, it is likely to be non-smooth, non-differentiable, and non-monotonic. As a result, the function's plot may be erratic, effectively reducing any GO method to an implicit brute-force search. Therefore, *goLoop* is only sound for SMT equations where the GO algorithm finds a minimum value of zero; otherwise, it is incomplete and unsound, as a zero-valued minimum may still exist at an unexplored point. While exhaustively evaluating all points would prove unsatisfiability, the vast input domain for FP or BV types – especially with multiple variables – makes this infeasible with current computing capabilities.

6 Conclusion and Outlook

We presented *goLoop*, a GO-based approach to model checking that handles terminating but variably bounded loops without explicit unrolling. By introducing an extended SMT syntax and translation rules for FP and BV theories, *goLoop* enables the direct execution of loops within the optimization process. This avoids the scalability limitations of BMC while maintaining compatibility with a wide range of derivative-free GO algorithms. Our presented examples – although not very complex – clearly illustrate the issues BMC encounters when handling loops with variable bounds, where *goLoop* could efficiently detect property violations by leveraging native execution of compiled code. These capabilities make it a complementary technique to BMC for verifying programs involving loops.

Future work will focus on developing an integrated tool based on the concepts of *goLoop* that enables the use of various derivative-free GO methods to automatically solve programs involving loops and constraints encoded in our proposed extended SMT-LIB2 syntax. This would enable a more sophisticated comparison between BMC tools and the *goLoop* approach to evaluate their efficiency in handling loop-intensive programs. Another aspect will be to analyze how the optimization function can be refined to improve the effectiveness of the applied GO methods. Furthermore, we plan to investigate how *goLoop* can be combined with existing SMT and BMC frameworks to support richer theories and composite system models. We also aim to explore parallel optimization strategies to further improve scalability for large and complex verification problems, potentially involving multiple nested loop constructs.

References

1. Alhawi, O.M., Rocha, H., Gadelha, M.R., Cordeiro, L.C., Batista, E.: Verification and Refutation of C Programs Based on K-Induction and Invariant Inference.

International Journal on Software Tools for Technology Transfer **23**(2), 115–135 (Apr 2021). https://doi.org/10.1007/s10009-020-00564-1

2. Ben Khadra, M.A., Stoffel, D., Kunz, W.: goSAT: Floating-point Satisfiability as Global Optimization. In: 2017 Formal Methods in Computer Aided Design (FMCAD). pp. 11–14. IEEE, Vienna (Oct 2017). https://doi.org/10.23919/FMCAD.2017.8102235
3. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic Model Checking without BDDs. In: Cleaveland, W.R. (ed.) Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
4. Clarke, E., Biere, A., Raimi, R., Zhu, Y.: Bounded Model Checking Using Satisfiability Solving. Formal Methods in System Design **19**(1), 7–34 (Jul 2001). https://doi.org/10.1023/A:1011276507260
5. Clarke, E., Kroening, D., Lerda, F.: A Tool for Checking ANSI-C Programs. In: Jensen, K., Podelski, A. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
6. Frohn, F., Giesl, J.: Integrating Loop Acceleration Into Bounded Model Checking. In: Platzer, A., Rozier, K.Y., Pradella, M., Rossi, M. (eds.) Formal Methods. pp. 73–91. Springer Nature Switzerland, Cham (2025)
7. Fu, Z., Su, Z.: XSat: A Fast Floating-Point Satisfiability Solver. In: Chaudhuri, S., Farzan, A. (eds.) Computer Aided Verification. pp. 187–209. Springer International Publishing, Cham (2016)
8. Grottke, M., Trivedi, K.S.: Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate. Computer **40**(2), 107–109 (2007). https://doi.org/10.1109/MC.2007.55
9. Güdemann, M.: Overview of Bounded Model Checking for Stack-Based Virtual Machines. In: Go Where the Bugs Are: Essays Dedicated to Wolfgang Reif on the Occasion of His 65th Birthday. Springer Nature Switzerland, Cham (2025). https://doi.org/10.1007/978-3-031-92196-4_9
10. Jeannet, B., Schrammel, P., Sankaranarayanan, S.: Abstract Acceleration of General Linear Loops. In: Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. p. 529–540. POPL '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2535838.2535843, https://doi.org/10.1145/2535838.2535843
11. Johnson, S.G., Schueller, J.: NLopt: Nonlinear Optimization Library. Astrophysics Source Code Library p. ascl:2111.004 (Nov 2021)
12. Khairullah, S.S., Elks, C.R.: Self-repairing Hardware Architecture for Safety-critical Cyber-physical-systems. IET Cyber-Physical Systems: Theory & Applications **5**(1), 92–99 (2020). https://doi.org/10.1049/iet-cps.2019.0022
13. Krahl, M., Güdemann, M., Wallentowitz, S.: parSAT: Parallel Solving of Floating-Point Satisfiability. In: Arusoaie, A., Cheval, H., Iosif, R. (eds.) Proceedings 9th edition of Working Formal Methods Symposium, Iaşi, Romania, 17 September 2025. Electronic Proceedings in Theoretical Computer Science, vol. 427, pp. 117–133. Open Publishing Association (2025). https://doi.org/10.4204/EPTCS.427.8
14. Kroening, D., Lewis, M., Weissenbacher, G.: Under-Approximating Loops in C Programs for Fast Counterexample Detection. Formal Methods in System Design **47**(1), 75–92 (Aug 2015). https://doi.org/10.1007/s10703-015-0228-1
15. Kroening, D., Sharygina, N., Tonetta, S., Tsitovich, A., Wintersteiger, C.M.: Loop Summarization Using State and Transition Invariants. Formal Methods in System Design **42**(3), 221–261 (Jun 2013). https://doi.org/10.1007/s10703-012-0176-y

16. Niemetz, A., Preiner, M.: Bitwuzla. In: Enea, C., Lal, A. (eds.) Computer Aided Verification. pp. 3–17. Lecture Notes in Computer Science, Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-37703-7_1
17. Solanki, M., Chatterjee, P., Lal, A., Roy, S.: Accelerated Bounded Model Checking Using Interpolation Based Summaries. In: Finkbeiner, B., Kovács, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 155–174. Springer Nature Switzerland, Cham (2024)
18. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**, 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2
19. Wu, G., Cao, W., Yao, Y., Wei, H., Chen, T., Ma, X.: LLM Meets Bounded Model Checking: Neuro-symbolic Loop Invariant Inference. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. pp. 406–417. ASE '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3691620.3695014
20. Xie, X., Chen, B., Zou, L., Liu, Y., Le, W., Li, X.: Automatic Loop Summarization via Path Dependency Analysis. IEEE Transactions on Software Engineering **45**(6), 537–557 (2019). https://doi.org/10.1109/TSE.2017.2788018

Probabilistic Relation-Changing Operators

Raul Fervari[1,2,3][0000−0003−0360−0725], Daniel Figueiredo[4,5][0000−0003−1727−9098], and Manuel A. Martins[4][0000−0002−5109−8066]

[1] FAMAF, Universidad Nacional de Córdoba, Argentina
[2] Consejo Nacional de Investigaciones Científicas y Técnicas, Argentina
[3] Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, France
[4] CIDMA, Department of Mathematics, Universidade de Aveiro, Portugal
[5] Association for Innovation and Biomedical Research on Light and Image, Coimbra, Portugal

Abstract. Relation-changing logics usually deal with dynamic phenomena over discrete models. In order to describe their properties and reason in this setting, different approaches were proposed, such as syntactical ones – where the model can be changed due to the use of some logical operator in the language – or the semantical ones – where the change in the system is parameterized by the model itself. The relation-changing paradigm has been studied and extended in the last years by considering, for instance, paraconsistent and fuzzy proposals. In this work, we aim to provide relation-changing logics over probabilistic transition systems. Precisely, we propose two relation-changing operators and explore how they affect the semantics, for instance by investigating adequate notions of probabilistic bisimulation.

Keywords: Relation-Changing · Probabilistic Logic · Probabilistic Kripke Model · Bisimulation · Sabotage Logic · Bridge Logic

1 Introduction

There are many kinds of probabilistic models to describe a probabilistic transition system. For instance, structures such as Markov Chains, a well-known example of the wider class of probabilistic automata (see e.g. [17]) play this role. Together with these models, we can use logical languages to describe features or reason about them. Different languages can be employed, such as the ones based on temporal logic (used, for instance, in PRISM to describe properties of Markov chains [13] and in [12] to describe knowledge) and probabilistic modal logics (see e.g. [1]), designed to reason about probabilistic Kripke structures.

A different concept associated with transition systems is that of relation-changing systems. In general, relation-changing systems are discrete state transition models whose accessibility relation may change after some event, such as crossing an edge. The exact way this configuration change is produced can vary from a semantical to a syntactical cause. For instance, we can consider models such as switch graphs, where the relation changes are encoded by the structure of the model itself [11]. In these cases, we can explore the model using classical

modal logic, without causing any change directly. Instead, updates are found as the result of crossing specific edges, according to the model specification. Another approach consists on considering a regular Kripke structure and explore it with a logic which embeds proper relation-changing operators on their syntax. These languages, such as sabotage and bridge logic [2,3,6], contain specific operators that while evaluating a formula, they cause the remaining subformula to be evaluated in a different model. In such a model, the accessibility relation may be different from the original one.

During the last years, the relation-changing paradigm has been implemented in different classes of models such as paraconsistent [5], weighted [10], fuzzy [7] and labeled [4,9] models. Also, some applications and computational implementations have also been developed. For instance, [10] presents a specific tool to deal with probabilistic cases, by considering very specific weights. Also, in [14], the same approach is employed in a practical context, showing its applicability for real-life problems. Other case-studies have been proposed, for instance, in [8,16]. This work aims to develop the study of this area by studying the introduction of relation-changing operators to probabilistic contexts, instead of considering preconceived models with limited expressiveness. While this has been done to models with fuzzy logic [7], probabilistic setting is not derivable as a particular case, since additional conditions must be imposed over the relation-changing process, in order to preserve the properties of a probabilistic model.

In this article we present a probabilistic approach for relation-changing logics, and study associated probabilistic bisimulation notions. In Section 2 we introduce the syntax, semantics and bisimulation notion for the basic probabilistic modal logic we consider. Section 3 is devoted to exploring probabilistic versions of sabotage and bridge modalities. Therein, we present their associated notions of bisimulations and show their respective invariance theorems. We finish in Section 4 with some remarks and future lines of work.

2 Probabilistic Modal Logic

2.1 Syntax and Semantics

In this section we present some preliminary notions over probabilistic logics. In what follows, let Prop be a countable set of atomic propositions. Also, for every relation E, we use the abbreviation $E[U] = \{v \mid (u, v) \in E \text{ for some } u \in U\}$.

Definition 1. *A* probabilistic Kripke model *is a tuple* $\mathcal{M} = (W, P, V)$ *where:*

- W *is a finite set of states or worlds;*
- $P : W \times W \to [0, 1]$ *is the transition probability matrix where for all* $w \in W$, $\sum_{w' \in W} P(w, w') = 1$*;*
- $V : \mathsf{Prop} \to 2^W$ *is a function that assigns to each* $p \in \mathsf{Prop}$ *the set of states in which* p *holds.*

We adopt the probabilistic definition of modal logic in [1]. Notice that the definition of truth is *crisp*, i.e., it is either true or false.

Definition 2 (Formulas). *The set of formulas is defined recursively as:*

$$\varphi, \psi ::= p \mid \neg\varphi \mid \varphi \vee \psi \mid \Diamond_\rho \varphi,$$

where $p \in \mathsf{Prop}$ *and* $\rho \in [0, 1]$.

Definition 3 (Satisfaction). *Let* $\mathcal{M} = (W, P, V)$ *be a probabilistic Kripke structure, and let* w *be a state in* W*. For any probabilistic modal logic formula* φ*, we define the satisfaction relation* $\models$ *recursively as follows:*

$$\begin{array}{lll} \mathcal{M}, w \models p & \textit{iff} & w \in V(p) \\ \mathcal{M}, w \models \neg\varphi & \textit{iff} & \mathcal{M}, w \not\models \varphi \\ \mathcal{M}, w \models \varphi \vee \psi & \textit{iff} & \mathcal{M}, w \models \varphi \textit{ or } \mathcal{M}, w \models \psi \\ \mathcal{M}, w \models \Diamond_\rho \varphi & \textit{iff} & \displaystyle\sum_{\{v \mid \mathcal{M}, v \models \varphi\}} P(w, v) \geq \rho. \end{array}$$

Example 1. To illustrate how the satisfaction is evaluated on a probabilistic Kripke model, consider the structure depicted in Fig. 1.

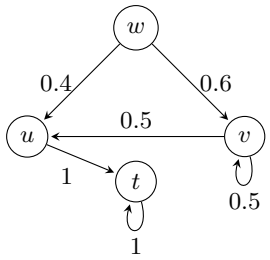


Fig. 1. Example of probabilistic Kripke model.

We define a model $\mathcal{M} = (W, P, V)$, where $W = \{u, v, w, t\}$, P is the transition probability matrix depicted in Table 1 and $V : \{p, q\} \to 2^W$ is defined by $V(p) = \{u, w, t\}$ and $V(q) = \{v, t\}$.

As example, let us evaluate the formula $\Diamond_{0.5}(p \vee \Diamond_{0.8}\ q)$ at the state w:

$$\mathcal{M}, w \models \Diamond_{0.5}(p \vee \Diamond_{0.8}\ q) \textit{ iff } \sum_{\{x \in W \mid \mathcal{M}, x \models p \vee \Diamond_{0.8}\ q\}} P(w, x) \geq 0.5.$$

Thus, for each state $x \in W$, we need to check whether $\mathcal{M}, x \models p \vee \Diamond_{0.8}\ q$, that holds if and only if $\mathcal{M}, x \models p$ or $\mathcal{M}, x \models \Diamond_{0.8}\ q$. Since $V(p) = \{u, w, t\}$, we only need to check the case $x = v$:

Table 1. Transition probability matrix of model depicted in Fig. 1.

	u	v	w	t
u				1
v	0.5	0.5		
w	0.4	0.6		
t				1

$\mathcal{M}, v \models \Diamond_{0.8}\ q$ *iff* $\sum_{\{y\ \in W\ \mid\ \mathcal{M}, y \models\ q\}} P(v, y) \geq 0.8$. According to the semantics, this is the case iff $P(v, v) + P(v, t) \geq 0.8$. But then we get $0.5 \geq 0.8$, that is false.

Hence, $\sum_{\{x \in W\ \mid\ \mathcal{M}, x \models p \vee \Diamond_{0.8}\ q\}} P(w, x) \geq 0.5$ holds iff $P(w, w) + P(w, u) + P(w, t) \geq 0.5$, iff $0.4 \geq 0.5$, which is false.

$\therefore\ \mathcal{M}, w \not\models \Diamond_{0.5}(p \vee \Diamond_{0.8}\ q)$.

2.2 Bisimulation

We now introduce a definition of bisimulation for the basic probabilistic case. While the classical definition usually defines bisimulation as an equivalence relation over the states of a probabilistic Kripke model, we introduce a version here to relate two different probabilistic Kripke models.

In [15], Panangaden studies probabilistic structures with labeled edges but without propositions. In that paper, the author introduces the definition of dynamic relation based on the work of Breugel & Worrell [19,20], which does not require the relation to be an equivalence relation. Moreover, he proves that a dynamic relation implies the existence of a bisimulation relation. There, he was also able to prove the invariance and Hennessy-Milner theorem for finite models. Despite that paper considers a probabilistic Kripke structure with labels, we use it as the basis for our definition within a context that also considers propositions.

Definition 4 (Bisimulation). *Let $\mathcal{M} = (W, P, V)$ and $\mathcal{M}' = (W', P', V')$ be two probabilistic Kripke models and $E \subseteq W \times W'$. We say that a nonempty relation E is a* bisimulation from $\mathcal{M}$ to $\mathcal{M}'$ *if for every $w \in W$ and $w' \in W'$ such that wEw', we have*

atoms: *for any $p \in \mathsf{Prop}$, $w \in V(p)$ iff $w' \in V'(p)$,*
zigzag$_{\mathbf{prob}}$: *for any $U \subseteq W$,* $\sum_{u \in U} P(w, u) \leq \sum_{u' \in E[U]} P'(w', u')$

With this definition at hand, we are able to prove the following characterization result, usually called the *invariance under bisimulation theorem.*

Theorem 1. *Let $\mathcal{M} = (W, P, V)$ and $\mathcal{M}' = (W', P', V')$ be two probabilistic Kripke models and $E \subseteq W \times W'$ a bisimulation from $\mathcal{M}$ to $\mathcal{M}'$. Then for any $(w, w') \in E$ and any formula φ we have*

$$\mathcal{M}, w \models \varphi \text{ iff } \mathcal{M}', w' \models \varphi$$

Proof. The proof is obtained as a fragment of the proof of Theorem 2, that will be presented later on the paper. □

3 Relation changing operators

This section presents our main contribution, which is defining novel logical languages which embed a relation-changing operator, interpreted over probabilistic Kripke structures. The languages extend those in e.g. [18,1]. The 'update' modality (written generally $\blacklozenge_\sigma$ here) will have two possible interpretations. First, as a sabotage-like modality (see [6,2,3]) in which the probability is subtracted from the edge accessing a chosen successor of the evaluation point, while it is transferred to another edge. Second, we introduce a bridge-like modality [6,2], in which the probability is instead taken from some edge to be added to the edge accessing the chosen successor.

The language we use is given by the following BNF:

$$\varphi, \psi ::= p \mid \neg\varphi \mid \varphi \vee \psi \mid \Diamond_\rho \varphi \mid \blacklozenge_\sigma \varphi,$$

where $p \in \mathsf{Prop}$, $\rho \in [0,1]$ and $\sigma \in]0,1[$. We denote the full set of formulas generated by the BNF above by Form. In the following, if the context is clear, we will use both $\Diamond_\rho$ and $\blacklozenge_\rho$.

The idea of changing a probabilistic relation implies that the transition probability matrix may change. We define $Prob(W)$, the set of all transition probability matrices over the set W as:

$$Prob(W) = \{P \in [0,1]^{W \times W} : \forall w \in W, \sum_{w' \in W} P(w,w') = 1\}.$$

3.1 Probabilistic Sabotage Operator

Let us start by introducing the semantics of a local sabotage modality. In [6,2], a local sabotage is interpreted as moving the evaluation of the formula to a successor of the current point, while removing the traversed edge. In the graded/fuzzy case [7], edges have associated values, so the sabotage is carried out instead by reducing the value of the traversed edge. Here, we will follow a similar approach, but maintaining the properties of a probabilistic Kripke model.

Definition 5. *Let $\mathcal{M} = (W, P, V)$ be a probabilistic Kripke structure, and let w be a state in W. For any formula $\varphi \in \mathsf{Form}$, we define the satisfaction relation $\models$ recursively as follows:*

$$\begin{array}{lll}
\mathcal{M}, w \models p & \textit{iff} & w \in V(p) \\
\mathcal{M}, w \models \neg\varphi & \textit{iff} & \mathcal{M}, w \not\models \varphi \\
\mathcal{M}, w \models \varphi \vee \psi & \textit{iff} & \mathcal{M}, w \models \varphi \textit{ or } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \Diamond_\rho \varphi & \textit{iff} & \sum_{\{v \mid \mathcal{M}, v \models \varphi\}} P(w,v) \geq \rho \\
\mathcal{M}, w \models \blacklozenge_\rho \varphi & \textit{iff} & \textit{exists } v \in W \textit{ s.t. } \rho \leq P(w,v) \neq 1, \textit{ and } \mathcal{M}_\rho^{-(w,v)}, v \models \varphi,
\end{array}$$

where $\mathcal{M}_\rho^{-(w,v)} = (W, P_\rho^{-(w,v)}, V)$*, with:*

- $P_\rho^{-(w,v)}(w,v) = P(w,v) - \rho$*;*
- $P_\rho^{-(w,v)}(w,t) = P(w,t) + \dfrac{\rho.P(w,t)}{1 - P(w,v)}$*, if* $t \neq v$*;*
- $P_\rho^{-(w,v)}(s,t) = P(s,t)$*, if* $s \neq w$*.*

Note that $\mathcal{M}_\rho^{-(w,v)}$ *is a well-formed model, since it is updated in the context of the precondition* $P(w,v) \geq \rho$ *and, according to its definition,* $P_\rho^{-(w,v)} \in Prob(W)$*.*

The semantic interpretation of the sabotage operator describes a scenario in which the probability of the event represented by an edge is reduced. When that is the case, the probability of the remaining ones (that also leave from the same origin state) is increased, in order to guarantee that the probability of all leaving edges sums to 1. For each of these edges, the increase is proportional to their pre-existing probability, in order to preserve the likelihood of each one being crossed. This is illustrated in Example 2.

Example 2. Let us consider the model (W, P, V) introduced in Example 1, and check if w satisfies $\blacklozenge_{0.4}p$. By the semantics we have $\mathcal{M}, w \models \blacklozenge_{0.4}p$ iff $\exists x \in W$ s.t. $0.4 \leq P(w,x) \neq 1$, and $\mathcal{M}_{0.4}^{-(w,x)}, x \models p$.

$P(w,x) \geq 0.4$ implies $x = u$ or $x = v$. Although $v \notin V(p)$, the formula is satisfied because $\mathcal{M}_{0.4}^{-(w,u)}, u \models p$. Moreover, the model $\mathcal{M}_{0.4}^{-(w,u)} = (W, P_{0.4}^{-(w,u)}, V)$ is illustrated in Fig. 2.

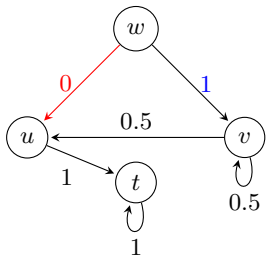


Fig. 2. The probabilistic Kripke model $(W, P_{0.4}^{-(w,u)}, V)$.

3.2 Probabilistic Bridge Operator

Now it is time to move to the bridge case, generalizing [6,2], and being more specific that in the graded case from [7]. The idea is that if there no way to access some point with a certain probability value, the evaluation is moved to such a point, and the value is added to the traversed edge. This is similar to the graded

case, but here we need to be careful to maintain the probability distribution. This is achieved by proportionally transferring the value from other edges to the intended one.

Definition 6. *The satisfiability relation* $\models$ *is defined as:*

$$\begin{array}{lll}
\mathcal{M}, w \models p & \textit{iff} & w \in V(p) \\
\mathcal{M}, w \models \neg\varphi & \textit{iff} & \mathcal{M}, w \not\models \varphi \\
\mathcal{M}, w \models \varphi \vee \psi & \textit{iff} & \mathcal{M}, w \models \varphi \textit{ or } \mathcal{M}, w \models \psi \\
\mathcal{M}, w \models \Diamond_\rho \varphi & \textit{iff} & \sum\limits_{\{v \mid \mathcal{M}, v \models \varphi\}} P(w, v) \geq \rho \\
\mathcal{M}, w \models \blacklozenge_\rho \varphi & \textit{iff} & \textit{exists } v \in W \textit{ s.t. } P(w, v) + \rho \leq 1, \textit{ and } \mathcal{M}_\rho^{+(w,v)}, v \models \varphi,
\end{array}$$

where $\mathcal{M}_\rho^{+(w,v)} = (W, P_\rho^{+(w,v)}, V)$*, with:*

- $P_\rho^{+(w,v)}(w, v) = P(w, v) + \rho$*;*
- $P_\rho^{+(w,v)}(w, t) = P(w, t) - \dfrac{\rho . P(w, t)}{1 - P(w, v)}$*, if* $t \neq v$*;*
- $P_\rho^{+(w,v)}(s, t) = P(s, t)$*, if* $s \neq w$*.*

Again, note that $\mathcal{M}_\rho^{+(w,v)}$ *is a well-formed model.*

The semantics of the bridge operator depicts a case where either the probability of an event to occur increases or some new transition is possible. Thus, the probability assigned to the other edges that also leave from the same origin state must decrease. As in the case of the "sabotage", this reduction is obtained in a proportional way, based on the previous value of each edge probability. This is illustrated in Example 3.

Example 3. Let us recall the model (W, P, V) from Example 1 and check if v satisfies $\blacklozenge_{0.4} q$. By the semantics we have $\mathcal{M}, v \models \blacklozenge_{0.4} q$ iff $\exists x \in W$ s.t. $0.4 \leq P(v, x) \neq 1$, and $\mathcal{M}_{0.4}^{+(v,x)}, x \models q$.

$P(v, x) \leq 0.4$ implies $x = w$ or $x = t$. Although $w \notin V(p)$, the formula is satisfied because $\mathcal{M}_{0.4}^{+(v,t)}, t \models p$. Moreover, the model $\mathcal{M}_{0.4}^{+(v,t)} = (W, P_{0.4}^{+(v,t)}, V)$ is illustrated in Fig. 3.

3.3 Bisimulation

Based on the Def. 4, we introduce a notion of bisimulation for the probabilistic language obtained by adding either the sabotage or bridge operator.

Definition 7 (Bisimulation). *Let* $\mathcal{M} = (W, P, V)$ *and* $M' = (W', P', V')$ *be two probabilistic Kripke models. We say that a non-empty relation* $E \subseteq \big(W \times Prob(W)\big) \times \big(W' \times Prob(W')\big)$ *is a* ρ*-sabotage (resp.* ρ*-bridge)* bisimulation from $\mathcal{M}$ to $\mathcal{M}'$ *if, for every pair* $\big((w, P), (w', P')\big) \in E$*, the following holds:*

atom: $w \in V(p)$ *iff* $w' \in V'(p)$*, for all* $p \in \mathsf{Prop}$*;*

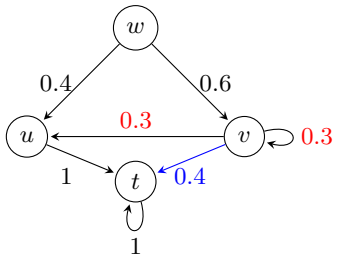


Fig. 3. The probabilistic Kripke model $(W, P_{0.4}^{+(v,t)}, V)$.

zigzag$_{\mathbf{prob}}$: *for any* $U \subseteq W$, $\sum_{u \in U} P(w,u) \leq \sum_{(u',P') \in \bigcup_{u \in U} E[(u,P)]} P'(w',u')$;

r−zig$_{\mathbf{prob}}$: *for any* $u \in W$ *s.t.* $1 \neq P(w,u) > \rho$ *(resp.* $P(w,u) + \rho < 1$*), exists* $u' \in W'$ *such that:*

1. $P(w,u) \leq P'(w',u') < 1$ *(resp.* $P(w,u) \geq P'(w',u')$*), and*
2. $\big(u, P_\rho^{-(w,u)}\big) E\big(u', P_\rho'^{-(w',u')}\big)$ *(resp.* $\big(u, P_\rho^{+(w,u)}\big) E\big(u', P_\rho'^{+(w',u')}\big)$*);*

r−zag$_{\mathbf{prob}}$: *for any* $u' \in W'$ *s.t.* $1 \neq P'(w',u') > \rho$ *(resp.* $P'(w',u') + \rho < 1$*), exists* $u \in W$ *such that:*

1. $P'(w',u') \leq P(w,u) < 1$ *(resp.* $P'(w',u') \geq P(w,u)$ *and*
2. $\big(u, P_\rho^{-(w,u)}\big) E\big(u', P_\rho'^{-(w',u')}\big)$ *(resp.* $\big(u, P_\rho^{+(w,u)}\big) E\big(u', P_\rho'^{+(w',u')}\big)$*).*

Moreover, for ρ*-bridge bisimulation, we also impose that for all* $w \in W, w' \in W'$, $E[\{(w,P)\}] \neq \emptyset \neq E^{-1}[\{(w',P')\}]$.

We say that E *is a sabotage bisimulation (resp. bridge bisimulation) if it is a* ρ*-sabotage (resp.* ρ*-bridge) bisimulation for every* $\rho \in [0,1]$*. Moreover, if* E *is a bisimulation and* $(w,P)E(w',P')$*, we say that* w *and* w' *are two sabotage (resp. bridge) bisimilar states.*

We note that our definition of sabotage bisimulation does not fully generalize the classical one. However, this is indeed caused by the semantical interpretation of the new operator $\blacklozenge_\rho$ that can distinguish two probabilistic Kripke models such as the ones illustrated in Fig. 4, which are bisimilar according to the classical definition. Thus, contrary to the classical case, we need to consider both **r−zig$_{\mathbf{prob}}$** and **r−zag$_{\mathbf{prob}}$** conditions. As it is possible to note in Fig. 4, the **r−zig$_{\mathbf{prob}}$** condition is verified from left to right model, but not in the opposite direction.

Theorem 2 (Invariance). *Let* $\mathcal{M} = (W,P,V)$ *and* $\mathcal{M}' = (W',P',V')$ *be two probabilistic Kripke models and let* $E \subseteq \big((W \times Prob(W)\big) \times \big(W' \times Prob(W')\big)$ *be a sabotage (resp. bridge) bisimulation. Then,* $(w,P)E(w',P')$ *implies* $\mathcal{M}, w \models \varphi$ *iff* $\mathcal{M}, w' \models \varphi$*, for every* $\varphi \in \mathsf{Form}$*, under sabotage (resp. bridge) semantics.*

Proof. We prove the theorem by induction over the structure of formulas. We firstly consider the sabotage semantics.

Base case:

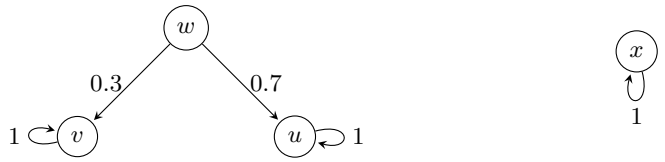


Fig. 4. Two model whose states are bisimilar but not sabotage/bridge bisimilar.

– If $\varphi \in \mathsf{Prop}$, then the result follows from **atom** from Def. 7.

Inductive cases:

– If $\varphi = \neg\psi$, then $\mathcal{M}, w \models \neg\psi$ iff $\mathcal{M}, w \not\models \psi$. By IH, $\mathcal{M}', w' \not\models \psi$, and then $\mathcal{M}', w' \models \neg\psi$.
– If $\varphi = \psi \vee \chi$, then $\mathcal{M}, w \models \psi \vee \chi$ iff $\mathcal{M}, w \models \psi$ or $\mathcal{M}, w \models \chi$. By IH, we get $\mathcal{M}', w' \models \psi$ or $\mathcal{M}', w' \models \chi$, hence $\mathcal{M}', w' \models \psi \vee \chi$.
– If $\varphi = \Diamond_\rho \psi$, then $\mathcal{M}, w \models \Diamond_\rho \psi$ iff $\sum_{(v \mid \mathcal{M}, v \models \varphi)} P(w, v) \geq \rho$. Note that (by Def. 7) $\sum_{\{v \mid \mathcal{M}, v \models \psi\}} P(w, v) \leq \sum_{\{v' \mid (v', P') \in \bigcup_{\{v \mid \mathcal{M}, v \models \psi\}} E[(v,P)]\}} P'(w', v') \leq \sum_{\{v' \mid \mathcal{M}', v' \models \psi\}} P'(w', v')$ (since $(v, P)E(v', P')$, and by IH). Moreover, by using the same reasoning we can conclude that $\sum_{\{v \mid \mathcal{M}, v \models \neg\psi\}} P(w, v) \leq \sum_{\{v' \mid \mathcal{M}', v' \models \neg\psi)} P'(w', v')$. Since for all $w \in W$, $\sum_{v \in W} P(w, v) = 1$ and for all $w' \in W'$, $\sum_{v' \in W'} P(w', v') = 1$, we get that $\sum_{\{v \mid \mathcal{M}, v \models \psi)} P(w, v) = \sum_{\{v' \mid \mathcal{M}', v' \models \psi\}} P'(w', v')$, because every $w \in W \cup W'$ satisfies either ψ or $\neg\psi$. Hence, we can conclude $\mathcal{M}, w \models \Diamond_\rho \psi$ iff $\mathcal{M}', w' \models \Diamond_\rho \psi$.
– If $\varphi = \blacklozenge_\rho \psi$, then $\mathcal{M}, w \models \blacklozenge_\rho \psi$ iff exists $v \in W$ s.t. $\rho \leq P(w, v) < 1$, and $\mathcal{M}_\rho^{-(w,v)}, v \models \psi$. From $\mathbf{r{-}zig_{prob}}$ in Def. 7, we conclude that exists $v' \in W'$ s.t. $P(w, v) \leq P'(w', v')$ and $(v, P_\rho^{-(w,v)})E(v', P_\rho'^{-(w',v')})$. Thus, by IH, exists $v' \in W'$, s.t. $\rho \leq P'(w', v') < 1$,and $\mathcal{M}_\rho'^{-(w',v')}, v' \models \psi$, and therefore $\mathcal{M}', w' \models \blacklozenge_\rho \psi$.
  The reciprocal implication is proved analogously, using $\mathbf{r{-}zag_{prob}}$.

Finally, for bridge semantics, the result is obtained using the analogous reasoning. Here we note that the additional condition that for all $w \in W$, $w' \in W'$, $E[\{(w, P)\}] \neq \emptyset \neq E^{-1}[\{(w', P')\}]$ ensures that it is not possible to create a bridge to a new state that was previously not bisimilar to any other. This is not required for sabotage because the probability assigned to edges with probability of 0 is never changed. □

Example 4. Let $\mathsf{Prop} = \{p\}$ and consider the two probabilistic Kripke models, as shown in Fig. 5. We denote the model on the left by $\mathcal{M} = (W, P, V)$ and the one on the right by $\mathcal{M}' = (W', P', V')$, with $V(p) = \{u, v, x\}$ and $V'(p) = \{u', v', x', y'\}$.

It is possible to verify that the following relation is a sabotage-bisimulation:
$E = \{\big((w, P), (w', P')\big), \big((z, P), (z', P')\big)\}$
$\quad \cup \{(t, P) \times (t', P') : \ t \in \{u, v, x\} \text{ and } t' \in \{u', v', x', y'\}\}$

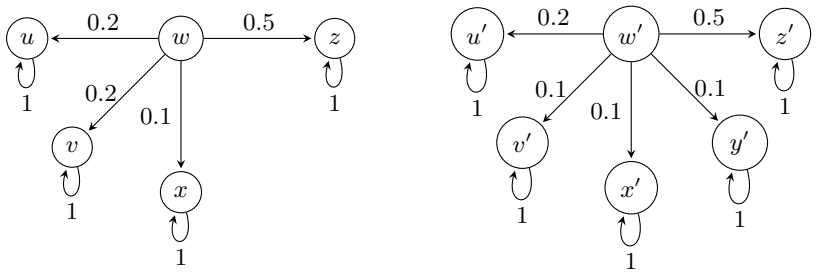


Fig. 5. Example of models with sabotage/bridge-bisimilar states.

$$\cup \{(z, P_\rho^{-(w,z)}) \times (z', P_\rho'^{-(w',z')}) : \ \rho \in]0, 0.5]\}$$
$$\cup \{(t, P_\rho^{-(w,t)}) \times (u', P_\rho'^{-(w',u')}) : \ t \in \{u, v\} \text{ and } \rho \in]0, 0.2]\}$$
$$\cup \{(x, P_\rho^{-(w,x)}) \times (t', P') : \ t' \in \{v', x', y'\} \text{ and } \rho \in]0, 0.1]\}$$

$\therefore$ w is bisimilar to w', z is bisimilar to z' and u, v, x are bisimilar to u', v', x', y'.

4 Conclusion and future work

In this paper, we extend the paradigm of relation-changing logics to probabilistic transition systems. While traditional relation-changing logics address dynamic phenomena over discrete models, we adapt this framework to settings where transitions are governed by probabilities. Specifically, we introduce two novel relation-changing operators tailored to the probabilistic case and analyze their impact on the underlying semantics. To support reasoning in this setting, we also investigate appropriate notions of probabilistic bisimulation, showing how these operators interact with key behavioral equivalences. This work thus opens a new direction in the study of relation-changing logics, complementing existing proposals in paraconsistent, fuzzy, and graded contexts [5,7].

This work is closely related to graded relation updates from [7], where classical relation-changing operators are extended to graded (fuzzy) relational structures. While that approach captures degree-based updates of accessibility, our proposal focuses on distributional updates over probabilistic transition systems, thus providing a complementary perspective on extending relation-changing logics beyond the classical setting.

As future work, we aim to develop model checking tools for relation-changing contexts. Although in a different context, rPRISM [10] – a tool which was integrated with PRISM – can be a starting point. We also would like to study the decidability status/complexity of the model checking problem and propose an automatic model-checker that integrates probabilistic sabotage and bridge operators.

Also, we would like to explore different semantics for relation-changing operators. In this approach, the changing operators are defined in such a way that

we move to a new accessible state (with an appropriate probability) after the change of the model. It would be interesting to study the case where we simply update the model without moving from the current state. A global version of these operators – where changes are not made locally but anywhere in the transition probability matrix of the model – would also be an interesting topic to explore.

Finally, we intend to explore the existence of a Hennessy-Milner theorem for finite models concerning our definition of bisimulation in the probabilistic setting, as well as in the model changing case.

Acknowledgments. D. Figueiredo and M. A. Martins acknowledge the support of CIDMA under the Portuguese Foundation for Science and Technology (FCT, https://ror.org/00snfqn58) Multi-Annual Financing Program for R&D Units, grants UID/4106/2025 and UID/PRR/4106/2025; and by FCT – Fundação para a Ciência e a Tecnologia through SACCCT- IC&DT - Sistema de Apoio à Criação de Conhecimento Científico e Tecnológico, as part of COMPETE2030, within the project BANKSY with reference number 15253. R. Fervari is supported by Agencia I+D+i grant PICT 2021-00400, the EU H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreements 101008233 (MISSION), the IRP SINFIN, SeCyT-UNC grant 33620230100178CB, and as part of France 2030 program ANR-11-IDEX-0003.

References

1. Alessandro Aldini, Gianluca Curzi, Pierluigi Graziani, and Mirko Tagliaferri. A probabilistic modal logic for context-aware trust based on evidence. *International Journal of Approximate Reasoning*, 169:109167, 2024.
2. Carlos Areces, Raul Fervari, and Guillaume Hoffmann. Relation-changing modal operators. *Logic Journal of the IGPL*, 23(4):601–627, 2015.
3. Guillaume Aucher, Johan van Benthem, and Davide Grossi. Modal logics of sabotage revisited. *Journal of Logic and Computation*, 28(2):269–303, 2018.
4. Suene Campos, Daniel Figueiredo, Manuel A. Martins, and Regivan Santiago. Labeled fuzzy reactive graphs. *Fuzzy Sets and Systems*, page 109320, 2025.
5. Diana Costa, Daniel Figueiredo, and Manuel A. Martins. Relation-changing models meet paraconsistency. *Journal of Logical and Algebraic Methods in Programming*, 133:100870, 2023.
6. Raul Fervari. *Relation-Changing Modal Logics*. PhD thesis, Universidad Nacional de Córdoba, 2014.
7. Raul Fervari, Daniel Figueiredo, and Manuel A. Martins. Graded relation updates in modal logic. In *Logic, Language, Information, and Computation - 31st International Workshop, WoLLIC 2025*, volume 15942 of *Lecture Notes in Computer Science*, pages 278–292. Springer, 2025.
8. Daniel Figueiredo and Luís Soares Barbosa. Reactive models for biological regulatory networks. In *International Symposium on Molecular Logic and Computational Synthetic Biology*, pages 74–88. Springer, 2018.
9. Daniel Figueiredo, Manuel A. Martins, and Luís S Barbosa. A note on reactive transitions and reo connectors. In *It's All About Coordination: Essays to Celebrate the Lifelong Scientific Achievements of Farhad Arbab*, pages 57–67. Springer, 2018.

10. Daniel Figueiredo, Eugénio Rocha, Manuel A. Martins, and Madalena Chaves. rPrism – a software for reactive weighted state transition models. In *Hybrid Systems Biology*, pages 165–174, Cham, 2019. Springer International Publishing.
11. Dov M Gabbay and Sérgio. Marcelino. Global view on reactivity: switch graphs and their logics. *Annals of Mathematics and Artificial Intelligence*, 66(1-4):131–162, 2012.
12. Joseph Y Halpern and Mark R Tuttle. Knowledge, probability, and adversaries. *Journal of the ACM (JACM)*, 40(4):917–960, 1993.
13. Marta Kwiatkowska, Gethin Norman, and David Parker. *Stochastic model checking*, pages 220–270. Springer, 2007.
14. Diogo Mendes, Daniel Figueiredo, Carlos Alves, Ana Penedones, Beatriz Costa, and Francisco Batel-Marques. Impact of the covid-19 pandemic on cancer screenings in portugal. *Cancer Epidemiology*, 88:102496, 2024.
15. Prakash Panangaden. Probabilistic bisimulation. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*, pages 290–322. Cambridge University Press, Cambridge, 2011.
16. Regivan Santiago, Manuel A. Martins, and Daniel Figueiredo. Introducing fuzzy reactive graphs: a simple application on biology. *Soft Computing*, 25(9):6759–6774, 2021.
17. Roberto Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.
18. Afsaneh Shirazi and Eyal Amir. Probabilistic modal logic. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007*, pages 489–495. AAAI Press, 2007.
19. Franck van Breugel and James Worrell. An algorithm for quantitative verification of probabilistic transition systems. In *CONCUR 2001 — Concurrency Theory*, pages 336–350, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
20. Franck van Breugel and James Worrell. Towards quantitative verification of probabilistic transition systems. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming*, pages 421–432, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

Reconfigurable stochastic multi-formalism models: an approach based on Maude

Lorenzo Capra[1] and Marco Gribaudo[2]

[1] Università degli Studi di Milano, Milano, Italy
[2] Politecnico di Milano, Milano, Italy

Abstract. Multiformalism modeling selects the best formalism for system components, while maintaining overall coherence. The escalating complexity and adaptability of systems necessitate the development of dynamic models to address them. We propose a framework based on `Maude` for reconfigurable multiformalism models. Two alternative solutions are presented. A server management case study using stochastic Petri nets and multiclass queuing networks showcases the feasibility of the approach. Experiments reveal that integrating rewriting techniques in multiformalism modeling potentially improves expressiveness and evaluation efficiency. Our short-term goal is to enhance SIMTHESys multiformalism software with dynamic transformations, utilizing `Maude` as the underlying rewriting engine.

Keywords: Multiformalism modeling · Rewriting systems · Model reconfiguration · Performance evaluation · Petri nets · Queuing networks

1 Introduction

Real systems require flexible modeling to be effectively evaluated, capturing critical features with appropriate tools and descriptions. Different aspects require different representations to minimize information loss. Specialists create tools that are tailored to their goals. For realistic systems, single-formalism models become impractical.

Multiformalism modeling evaluates system properties by integrating diverse techniques to address both qualitative and, in particular, quantitative aspects. It allows selecting suitable formalisms for each system component, enhancing expressiveness and interpretability while maintaining unity. Effective analysis of complex models requires coordinated strategies. Advanced methodologies use composability to tackle modeling complexities, employing flexible tools like decomposition and symbolic techniques to handle challenges like submodel misalignment and state-space explosion.

However, the increasing (self-)adaptability and reconfiguration requirements of modern distributed systems necessitate the development of dynamic multiformalism models to address them. We here propose a modular framework based on `Maude`, a purely declarative language with Rewriting Logic sematics, for *reconfigurable* multiformalism models: We present and discuss two alternative solutions,

using a server management case study (integrating stochastic Petri nets and multiclass queuing networks) to showcase the feasibility of the approach. We also provide some experimental data.

Section 2 discusses related work; Section 3 recalls the main features of `Maude`; Section 4 presents the case study; the core Section 5 introduces a `Maude`-based framework for reconfigurable multiformalism models using the case study as an illustrative example; two alternative solutions are proposed; a few experimental data for the case study analysis are given in Section 6; Finally, Section 7 concludes the paper and outlines the ongoing work.

2 Related work

Multiformalism modeling [13][16], often paired with *multisolution*, is well-studied in fixed combinations, such as DEDS [3], SMART [9] and SHARPE [17], and in dynamic combinations, such as AToM^3 [15] and Möbius [11]. This paper's research may relate to OsMoSys [12] and SIMTHESys [14][1], which use a multisolution approach. These systems, along with Möbius, create an optimized executable model from a descriptive representation that includes behaviors of programmatically defined model elements. OsMoSys solves models by orchestrating workflows that activate external *solvers* based on intermediate submodel results. SIMTHESys separates model descriptions from model elements, allowing it to define interformalism interactions and provide analysis tools. This enables automatic creation of multiformalism *solvers* for models based on consistent formalisms. This feature allows defining new formalisms and their combinations, enabling users to experiment and develop optimized tools with minimal software development if successful. [2] presents an overview of the characteristics and limits of these tools.

3 The `Maude` System

`Maude` [10] is an expressive, high-performance, purely declarative language with rewriting logic semantics [4]. The `Maude` runtime provides various facilities for model checking, verification of LTL formulae, and symbolic reachability. `Maude` has served as a logical framework for various other formalisms, including Petri nets, Automata, and Process Algebras, which, despite their strength, do not possess the essential features needed for intuitively defining adaptable systems.

`Maude` syntax is based on *equations* and *rules*. Each side of a rule or equation is a *term* of a certain *kind*, which may involve variables. Rules and equations operate through intuitive rewriting, where instances on the left side are replaced with instances on the right.

A *functional* module defines operations using equations as simplifications. It outlines a *equational theory* $(\Sigma, E \cup A)$ of membership equational logic : Σ is the signature, which includes the declaration of *sorts*, *subsorts*, *kinds*[3] and

[3] are implicit equivalence classes formed by connected components of sorts under the subsort partial order, and terms of a kind without a sort denote *errors*.

operators; E contains equations and membership axioms; and A contains the equational attributes of operators (e.g., assoc, *comm*, *ide*). The mathematical model of $(\Sigma, E \cup A)$ is the *initial algebra* $T_{\Sigma/E\cup A}$, formed by the equivalence classes of the relation induced by $E \cup A$ in the ground-term algebra T_Σ. Under the conditions (modulo-A) of confluence, subsort decreasing, and termination on (Σ, E), any ground term is rewritten in a unique canonical form that has the least sort in the subsort PO: The canonical term algebra is isomorphic to the initial algebra, which makes the denotational and operational semantics consistent.

A *system* module includes *rewrite rules* that represent local transitions in a concurrent system. It defines a *rewrite theory* [4] $\mathcal{R} = (\Sigma, E \cup A, R)$. Here, $(\Sigma, E \cup A)$ acts as the underlying equational theory, and R is a set of rewriting rules. This theory captures the behavior of a concurrent system, with $(\Sigma, E \cup A)$ defining the algebraic structure of the states and R describing the concurrent transitions. The initial model of $\mathcal{R}$ provides each kind k with a labeled transition system (TS) where states are elements of $T_{\Sigma/E\cup A,k}$ and state transitions occur as $[t] \xrightarrow{[\alpha]} [t']$, with $[\alpha]$ denoting an equivalence class of rewrites. The *coherence* property [10] ensures that a strategy reducing terms to canonical forms before applying rules (adopted by the `Maude` rewrite engine) is sound and complete, if a matching-modulo-A algorithm is used.

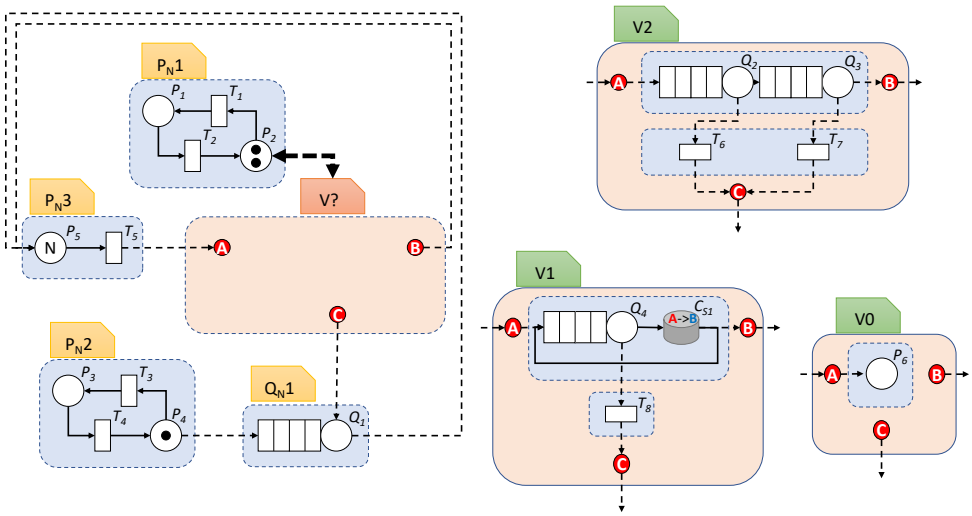


Fig. 1. Multiformalism model of the case-study: on the left, the entire system (V?: reconfigurable server sub-system; Q_{N_1} backup single server; P_N1 and P_N2: fault-repair modulating processes; P_N3: request injection process); on the right, the three possible server-reconfiguration scenarios modulated by P_N1.

4 Case Study

The case study consists of a server that encompasses two computing facilities, both of which are susceptible to faults and capable of undergoing repairs. This is an advance of the example presented in [14], where the implementation within SIMTHESys is elucidated in terms of formalism definitions. The facilities possess buffers for incoming requests and cease to accept further requests upon buffer depletion. A front-end system allocates requests to the facilities, allowing the second facility to undertake jobs initially directed to the first, provided that they are not processed before a timeout occurs. Requests are executed in two sequential stages, which, under optimal operating conditions, are performed by two cooperatively functioning servers. To manage periods of high request arrival rates, any request that resides in the queue beyond a predetermined threshold is diverted to a third backup server. All three nodes are periodically unavailable due to scheduled maintenance. In instances where one node in the tandem system becomes unavailable, the remaining node assumes responsibility for both stages of processing. The system is represented through three stochastic Petri Nets (SPN) submodels, denoted P_N1, P_N2, and P_N3, which address fault/repair processes and request injection. Additionally, two Multiclass Queuing Networks (MQN) submodels, with exponentially distributed service times (Q_N1 and $V?$), are employed for the three service nodes (refer to Fig. 1).

Multiformalism interactions among SIMTHESys submodels are realized utilizing *SIMTHESys bridges*, which can interact comprehensively with the MQN and SPN elements. Specifically, the arc directed from the SPN place P_4 to the MQN queue Q_1 serves as an enabling arc (a self-loop in SPN). Furthermore, the arc from an SPN transition to an MQN queue functions like an output arc by enqueuing a task upon the SPN transition firing. In contrast, arcs from an MQN queue (designated as Q_1, Q_3, or C_{S1}) to the SPN place P_5 generate a token within the place upon completion of the queue job processing. Additionally, arcs going from an MQN queue to an SPN transition ($Q_2 \to T_6$, $Q_3 \to T_7$, or $Q_4 \to T_8$) serve as an extension of the transition enabling in SPN, subtracting a task from the queue upon their activation.

The figure's left section illustrates the primary model: submodel P_N3, which consists of place P_5 and transition T_5, is responsible for defining the arrivals of requests to the system. The principal service is represented by the rewritable model indicated as $V?$, while the backup node comprises a submodel, Q_N1, which includes the singular queue Q_{N_1}. The subsystems are regulated by the Petri net models P_N1 and P_N2 (covered by straightforward P-semiflows, that is, conservative): the backup node exhibits an On-Off behavior, as described by the sub-models P_N2 and Q_N1, while the primary subsystem is managed by a basic Petri net characterized by two places and two tokens (P_N1). The replacement of the component $V?$ with $V2$, $V1$, or $V0$, as depicted on the right side of the figure, depends on the token count in place P_2.

If both servers are available (two tokens in P_2), the submodel $V2$ is used, where each server is represented by an individual queue (Q_2 and Q_3). If one node is down (one token in P_2), the submodel is replaced by the single queueing

station Q_4 (submodel $V1$): the stages become two classes of customers. If a job in the first stage finishes, it enters the class switch (C_{S1}) and reenters the server with the class representing the second stage; otherwise, it leaves the system. If both nodes are down, no service is performed, and the system is stopped. In this case, the queue is replaced by a place (P_6 of the submodel $V0$), whose marking encodes the current stage of the corresponding job.

5 Using `Maude` as a framework for reconfigurable multi-formalism models

In this section, we discuss the use of `Maude` (and its runtime support) as a rewriting engine for multi-formalism models. The advantages of this choice are multiple: flexibility and modularity of the modeling, efficiency, soundness, ease of integration with existing tools, and upgrade. We inherit the `Maude` facilities for formal verification; in addition, we can exploit a new quantitative analysis capability based on the association of a Markov process to `Maude` executable modules, introduced in [8,7] for rewritable SPN and fully developed in [6].

Though our framework is off-the-shelf, rewriting engines allow for dynamic submodel modifications (OsMoSys *late binding* feature), integrating into SIMTHESys multiformalism solvers. In perspective, this could enable model rewriting at *solution* time, simplifying and streamlining the solution process, and making model design easier and more accessible.

After intuitively describing the general approach to multiformalism modeling using `Maude`, we will instantiate it in the case study.

We skip some technical details and refer to `https://github.com/lgcapra/rewpt/tree/main/multiformalism` for a complete description. However, we include a few code excerpts for the reader's convenience.

5.1 Reconfigurable multiformalism models in `Maude`

The multiformalism encoding in `Maude` is based on a simple, compact, and extensible hierarchy of modules, some of which are *parametrized*. We propose two solutions, one characterized by simplicity and the other more elegant and concise, though based on advanced features of `Maude` module operations. In the following description, we focus on the former. Our aim is to reuse existing modules (that encode particular formalisms) for convenience.

A parametrized module uses type parameters expressed by (functional) *theories*. These theories establish module interfaces by setting syntactic and semantic properties for parameter modules. They are similar to functional modules but are not required to be Church-Rosser or terminating. Theories have loose semantics, allowing any algebra that meets the equations and membership axioms to be acceptable. In `Maude`, *views* link a source theory to a target module or theory, detailing the mapping of sorts and operators.

Solution 1 As usual, the formalization is based on multisets, which are implemented both as a commutative monoid built on a set or in a more compact way as weighted sums (module `BAG{X :: TRIV}`), for efficiency reasons.

Both solutions rely upon a fundamental concept: a multiformalism model is constructed by integrating multiple heterogeneous model components (e.g., Petri nets, queue networks, BPMN, activity diagrams, process algebra, etc.) that collectively maintain a notion of distributed state. This notion is formalized by the theory `STATE` (see the excerpt below), which requires two *sorts* `State`, `LocState` linked by the *subsort* relationship (corresponding to the set $\subseteq$ relation) and an associative-commutative (AC) juxtaposition operation. This theory describes the parameter of the functional module `NETWORK{L :: STATE}`, which provides the simple Abstract Data Type (ADT) of a multiformalism model.

The abstract structure of the model is defined by the sort `Network` and the AC juxtaposition `_,_` having the ground term `emptyNetW` as identity. In other words, a multiformalism model is a multiset (i.e. a commutative monoid) of possibly different `Network` components, called nodes in the following. The idea is straightforward: by suitably "instantiating" the several node types through specific subsort relationships, we can build a model consisting of arbitrary concrete components interacting through shared state elements (called *places*).

A term of sort `NetSys` is made up of a `Network` and a `State`, and describes a network of connected nodes with a distributed associated *state*.

In our example, the network nodes use a notion of state matching that of a Petri net's *marking*, that is, a multiset of places compactly represented using the `PBAG{PL ::TRIV}` module (the trivial theory `TRIV` only contains the declaration of sort `Elt`). The type-parameter of this module indicates the place label: we will use places indexed by naturals. Having a flexible place labeling is important, as we will explain later. The AC operator `_+_`, which is also marked as a constructor, provides a description of multisets as weighted sums: For example, the term `3. p(1) + 1. p(2)` of sort `Pbag` (from the module instantiation `PBAG{Nat}`), represents a multiset with three occurrences of p_1 and one of p_2.

The parametrized view `S-Pag{PL :: TRIV}` is used to instantiate the theory `STATE` as a multi-set of places via an appropriate mapping. Specifically, the fundamental element of a state is delineated by the sort `ElPbag`, which encapsulates elementary bags such as `3 . p(1)`. Using this view, we derive the ADT of heterogeneous networks that incorporate the distributed state representation of the marking, as defined in the module `NETWORK-MARKING{PL :: TRIV}`.

Listing 1.1. Multiformalism definition: solution 1 1

```
fth STATE is
   sorts State LocState .
   subsort LocState < State .
   op _+_ : State State -> State [assoc comm] .
endfth

fmod NETWORK{S :: STATE} is
 protecting EXT-BOOL .
 sorts Network NetSys .
 op emptyNetW : -> Network [ctor] .
 op _,_ : Network Network -> Network [ctor assoc comm prec 123 id: emptyNetW] .
 op _:_ : Network S$State -> NetSys [ctor prec 125] .
 op netw : NetSys -> Network .
 op state : NetSys -> S$State .
 vars N N' : Network .
 var M : S$State .
 eq netw((N : M)) = N .
 eq state((N : M)) = M .
endfm

view S-Pbag{PL :: TRIV} from STATE to PBAG{PL} is
 sort State to Pbag .
 sort LocState to ElPbag .
endv

fmod NETWORK-MARKING{PL :: TRIV} is
  protecting NETWORK{S-Pbag{PL}} .
endfm
```

Components used in the case-study Two types of nodes are used: Stochastic Petri Nets (SPN) and Multi-class Queue Networks (MQN). The functional modules describing their signature are not included for space reasons. For the former type, we largely reuse the signature of (rewritable) SPN [7], which is based, in turn, on that of (rewritable) Place-Transition (PT) nets with inhibitor edges provided in [5,8]. The SPN formalization includes a small hierarchy of modules available at `https://github.com/lgcapra/rewpt`.

SPN The functional module `SPN-NODE{PL :: TRIV}` describes the SPN nodes: imports `NETWORK-MARKING` and the predefined module `SPN-SIG` (parametrized on both place and transition labels), using the type parameter `PL` as an actual parameter in the imported modules. Importing `SPN-SIG` with the `protecting` mode intuitively means that its initial semantic is retained, while import `NETWORK-MARKING` with the `extending` mode means that the sort `Network` will be extended with new data values, without identifying previously defined ones.

SPN transitions (terms of sort `Tran`) are described through labels linked to adjacency lists, expressed as `Pbag` triples `[I, O, H]`. These labels include a

descriptive tag (a `String` in our example), a `Float` (the rate parameter of a negative exponential firing delay) and a `Nat` (which specifies the firing policy). For instance, the ground term,

```
t("a", 1.5, 0) |-> [1 . p(1) + 2 . p(2), 1 . p(1), 2 . p(1)]
```

delineates a transition characterized by the label "a", an exponential firing rate $\mu = 1.5$, and an infinite-server type. This transition requires the presence of precisely one token in place p_1 and a minimum of two tokens in place p_2 to be enabled. Upon firing, it will remove two tokens from p_2. The PT net underlying an SPN (a term of sort `Net`) is straightforwardly defined in a modular way using the AC juxtaposition `;` and the subsort relationship `Tran < Net`.

Using the predefined `firingRate` operator, we can define marking-dependent rates: The current definition is based on the enabling degree ($ed(t, m)$), which refers to the occurrences of a transition that are simultaneously enabled in a marking. Under the infinite server policy (0), the transition exponential rate is $\mu \cdot ed(t, m)$. Under the k-server policy, $k > 0$, it is $\mu \cdot min(ed(t, m), k)$.

To reuse SPN transitions within a multiformalism framework, it is necessary to encapsulate the SPN signature (referred to as the `SPN-SIG` module) in conjunction with the ADT of the network (the `NETWORK` module) into a new functional module (`SPN-NODE`). This module should incorporate the subsort relationship `Tran < Network`. Alternatively, the relationship `Net < Network` could be established, by which both the expressions `(t1 ; t2), t3` and `t1, t2, t3` are acknowledged as terms of the `Network`, although different.

Using this approach, we can add new types of nodes in a standardized way.

Multi-class Queue Networks MQN are straightforwardly built from elementary queues (representing single severs), that is, terms of sort `ElQueue`, as defined in the module `EL-QUEUE{PL :: TRIV}`. This term comprises a `Place` and a `Float` (the exponential service rate), represented by the notation `P @ F`.

The MQN signature is defined in module `QUEUE{PL :: TRIV}` (which includes the predefined parametrized modules `LIST` and `PBAG` in `protecting` mode). A simple MQN is a term of sort `SimpleQ` obtained by juxtaposing a non-empty list of elementary queues (a term of sort `NeList{ElQueue}`, obtained by instantiating `LIST` via a parametrized view) followed by a `Place`, which represents the MQN endpoint: the used notation is `NeL > P`. The general form of an MQN, identified as a term of sort `Queue`, consists of a `SimpleQ` that is prefixed by two `Pbag` terms enclosed between `[]`: These multisets represent the queue's input and inhibitor conditions relative to the surrounding context. The intuitive subsort relationship `SimpleQ < Queue` is established.

With this notation, it is possible to flexibly represent any MQN. For example, the `SimpleQ` term `p(1) @ 1.0 p(2) @ 1.5 > p(3)` represents a two-class queue, while the `Queue` term `[2 . p(1), nilP] p(4) @ 2.0 > p(5)` denotes a single-class queue whose enabling requires two tokens in place `p(1)`. The connection of MQN nodes to a heterogeneous network is simply achieved (analogously to the SPN nodes) through the parametrized module `QUEUE-NODE`, which wraps

the NETWORK-MARKING ADT of nodes using the distributed state notion of marking and the MQN signature (QUEUE).

Arbitrary combinations can be formed by composing network nodes (,): For instance, if q1 and q2 are Queue terms with the endpoint of q1 aligning with the starting place in q2, then q1, q2 simply represents their sequence (note that its time semantics is different from a single MQN that incorporates q1 and q2).

Node dynamics The behavior of a multiformalism model is characterized by system modules, each associated with a specific type of node. In our example, we have the parametrized modules SPN-NODE-SYS and QUEUE-NODE-SYS (listing 1.2): These modules encompass the conditional rewrite rules that specify the (timed) semantics of the nodes within a network. These rules locally modify the state (marking) of a SysNet term N : M, which represents a network of interconnected nodes with an associated multiset of places, due to the occurrence of an SPN transition or the execution of the service for a client in an MQN, that either reaches the next server or exits the MQN. Leaving out technical details, we point out that the free variable rate is bound (through a matching equation :=) to an expression that accurately defines the time semantics for the corresponding event. Observe that in the case of an MQN, this depends on the ratio of clients located at a specific place relative to the total population of the MQN. This representation facilitates the automated derivation of the CTMC generator matrix, as detailed in [6].

The following excerpt illustrates the link of a specific node type (SPN) to the ADT of the network using a standard procedure.

Listing 1.2. Linking a node type to the Network ADT

```
fmod SPN-NODE{PL :: TRIV} is
  extending NETWORK-MARKING{PL} .
  protecting SPN-SIG{String, PL} .
  subsort Tran < Network .
  *** subsort Net < Network . *** alternative
endfm

mod SPN-NODE-SYS{PL :: TRIV} is
  including SPN-NODE{PL} .
  var N : Network . var T : Tran . vars B B' : Pbag . var K : NzNat . var rate : Float .
  crl [spn-t] : (N , T) : B => (N , T) : B' if enabled(T, B) /\ B' := firing(T, B) /\
       rate := firingRate(T, B) . *** PN firing rule
endm
```

Case study's model The encoding of the multiformalism model depicted in Figure 1 is systematically presented in listing 1.3. The associated system module (MQN-SPN) comprehensively integrates the two system modules that delineate the dynamics of the nodes. In addition, it specifies the rules (with labels $[V_i > V_j]$) that govern the structural transformation of a component of the network (indicated by V_i), depending on the marking of the place p_2, as illustrated in the description of the case study.

Listing 1.3. Case study formalization including network reconfiguration

```
mod MQN-SPN is
  inc SPN-NODE-SYS{Nat} .
  inc QUEUE-NODE-SYS{Nat} .
  var K : NzNat . *** model parameter
  vars N N' N'' : Network . var S : Pbag .
  ops t0 t1 t2 t3 t4 t5 t6 : -> Tran . *** aliases
  ops eq1 eq2 eq3 : -> ElQueue . *** aliases
  eq eq1 = p(7) @ 1.0 . *** conditioned single-class queue
  eq eq2 = p(1) @ 1.5 .
  eq eq3 = p(6) @ 2.5 .
  ops q1 q2 q3 q23 : -> Queue . *** aliases
  eq q1 = [1 . p(5), nilP] eq1 > p(0) .
  eq q2 = [2 . p(2), nilP] eq2 > p(6) .
  eq q3 = [2 . p(2), nilP] eq3 > p(0) .
  eq q23 = [1 . p(2), nilP] ql(q2) ql(q3) > out(q3) . *** MQN alias
  op network : -> Network . *** alias
  op netsys : NzNat -> NetSys . *** alias
  eq t0 = t("start", 1.0, 1 ) |-> [1 . p(0), 1 . p(1), nilP] .
  eq t1 = t("switch1", 0.5, 1 ) |-> [1 . p(3), 1 . p(2), nilP] .
  eq t2 = t("switch2", 0.05, 1 ) |-> [1 . p(2), 1 . p(3), nilP] .
  eq t3 = t("on", 2.0, 1 ) |-> [1 . p(4), 1 . p(5), nilP] .
  eq t4 = t("off", 1.0, 1 ) |-> [1 . p(5), 1 . p(4), nilP] .
  eq t5 = t("rem1", 1.0, 1 ) |-> [1 . p(1), 1 . p(7), nilP] .
  eq t6 = t("rem6", 1.5, 1 ) |-> [1 . p(6), 1 . p(7), nilP] .
  op V : NzNat -> [Network] [memo] . *** variable component (depends on p(2))
  eq V(2) = q2 , q3 , t5 , t6 . *** sequential composition: "out" of q2 is "in" for q3
  eq V(1) = q23 , t6 . *** hybrid component (MQN + tran)
  eq V(0) = p(eq2) @ 0.0 p(eq3) @ 0.0 > out(q3) . *** "dead" queue
  eq network = t0 , t1 , t2 , t3 , t4 , t5 , t6 , q1 , V(2) .
  eq netsys(K) = network : K . p(0) + 2 . p(2) + 1 . p(5) .
  *** network rewriting
  crl [V2>V1] : N : S => N' , V(1) : S if S[p(2)] = 1 /\ N'' , N' := N /\ N'' = V(2) .
  crl [V2>V0] : N : S => N' , V(0) : S if S[p(2)] = 0 /\ N'' , N' := N /\ N'' = V(2) .
  crl [V1>V2] : N : S => N' , V(2) : S if S[p(2)] = 2 /\ N'' , N' := N /\ N'' = V(1) .
  crl [V1>V0] : N : S => N' , V(0) : S if S[p(2)] = 0 /\ N'' , N' := N /\ N'' = V(1) .
  crl [V0>V2] : N : S => N' , V(2) : S if S[p(2)] = 2 /\ N'' , N' := N /\ N'' = V(0) .
endm
```

For example, the rule with the label `V2 > V1` denotes the replacement of component V_2 with V_1 when p_2 has one token. This methodology represents a conventional approach to construct a multiformalism model composed of various types of reconfigurable components.

Rewrite rules must exhibit coherence with equations and pattern-matching modulo A, as elucidated at the conclusion of section 5. Given that the `Maude` rewriting engine operates under the assumption of coherence without verification, meticulous attention is necessary. For instance, the rudimentary formulation of the rule $[V2 > V1]$ is inadequate.

```
crl [V2>V1] : N , V(2) : S => N , V(1) : S if S[p(2)] = 1 .
```

In accordance with the coherence assumption, the equation corresponding to the term `V(2)` (utilized as an alias) is anticipated to be applied initially. Consequently, this results in the right-hand side of the rewrite rules remaining unmatched. A detailed discussion of this important topic is beyond the scope of this paper. Under the executability assumption in the equational theory $E \cup A$, coherence in the multiformalism framework is ensured if rules' right-hand sides use only variables and constructors (possibly non-free) at each level.

Solution 2 We propose an alternative framework for multiformalism modeling in `Maude`. The objective is to improve the characterization of the nodes within a heterogeneous network, thus facilitating the modeler's efforts. Conceptually, these nodes are interconnected through a shared notion of distributed state, and they possess an intrinsic semantics that induces local state transformations. Implicit in the previous formalization, this concept is now made explicit and encapsulated in the theory `NODE`, subsequently extending the theory `STATE`.

The leading element of the theory `NODE` is an operator, denoted `next`, which encapsulates the reachability relationship between the internal states of a node. The sort `StateRate`, equipped with appropriate getters, constitutes a pair comprising a reachable state and a floating-point number: hence, given a node alongside its current state, this operator yields a multiset that represents the potential one-step transitions from the node's current state to novel local states, along with the associated rates. This multiset is aligned with the sort `StatesRates`, customarily defined as a free commutative monoid. In essence, this operator represents the potentially non-deterministic firing rule operative for each network node. A nuanced justification for employing a multiset instead of a set arises from the potential existence of multiple transitions from a singular source state to an identical target, a factor that, as expounded in [7,6], must be properly considered to precisely calculate the stochastic matrix of a model. It should be noted that this methodology may be readily extended to account for structural reconfigurations of nodes, for example, by utilizing multisets of triplets to represent the novel configurations (comprising node and state) alongside their transition rates.

The module `NETWORK-SIG`, which resembles `NETWORK{S :: STATE}` as presented in Solution 1, delineates the abstract signature of the network. The difference is that it is not parameterized and introduces the `State` sort. The functional module `NETWORK-NODE{N :: NODE}`, which imports `NETWORK-SIG` in a `protecting` mode, syntactically associates a family of nodes of a specified type with the network through the subsort relationship `N$State < State` and `N$Node < Network{N} < Network`. Some operator overloading facilitates the recognition of homogeneous (sub)networks.

The system module `NETWORK-SYS{N :: NODE}` includes `NETWORK-NODE{N}` and defines in a systematic way the rewrite rule that encodes inner state transitions using the operator `next` required by the theory `NODE`. The remaining part, which depends on the specific model, is available at (`https://github.com/lgcapra/rewpt/tree/main/multiformalism/NETWORK-2.maude`).

This part covers the implementation of a multiset of state transitions, inclusive of their respective rates, along with two functional modules corresponding to each node type: `SPN-SIG+` and `QUEUE+`. These modules are parameterized with respect to the node labels and import the signatures of SPN and MQN, respectively, under a modality `protecting`. Each module provides a definition of the `next` operator. The implementation for MQN demonstrates non-determinism because many local-state transitions may occur within a given MQN.

The system module that formalizes the case study is analogous to the one in listing 1.3, except for the integration of two instances of `NETWORK-SYS{N :: NODE}` through two parametrized views that link the parameter `N` to `SPN-SIG+` and `QUEUE+`, respectively.

Listing 1.4. Multiformalism definition: solution 2

```
fth NODE is
  inc STATE . *** includes another theory
  pr FLOAT .
  sort Node .
  sort StateRate . *** a pair (state,rate)
  sort StatesRates . *** multiset of pairs (state,rate)
  subsort StateRate < StatesRates .
  op state : StateRate -> State .
  op rate : StateRate -> Float .
  op empty : -> StatesRates .
  op __ : StatesRates StatesRates -> StatesRates [assoc comm id: empty] .
  op next : Node State -> StatesRates . *** encodes the firing rule
endfth

fmod NETWORK-SIG is
   sorts Network State NetSys .
   op emptyNetW : -> Network [ctor] .
   op _,_ : Network Network -> Network [ctor assoc comm prec 123 id: emptyNetW] .
   op _:_ : Network State -> NetSys [ctor prec 125] .
   op netw : NetSys -> Network .
   op state : NetSys -> State .
   var N : Network . var M : State .
   eq netw((N : M)) = N .
   eq state((N : M)) = M .
endfm

fmod NETWORK-NODE{N :: NODE} is
  pr NETWORK-SIG .
  sorts Network{N} NetSys{N} .
  subsort N$Node < Network{N} < Network .
  subsort N$State < State .
  subsort NetSys{N} < NetSys .
  op _,_ : Network{N} Network{N} -> Network{N} [ctor ditto] . *** overloading
  op _:_ : Network{N} N$State -> NetSys{N} [ctor ditto] .
endfm
```

```
mod NETWORK-SYS{N :: NODE} is
  inc NETWORK-NODE{N} .
  var N : N$Node . vars M M' : N$State . var SR : N$StateRate .
  var SRs : N$StatesRates . var NW : Network . var R : Float .
  crl [firing-rule] : (N, NW) : M => (N, NW) : M' if SR SRs := next(N, M) /\ M' :=
       state(SR) /\ R := rate(SR) .
endm
```

5.2 Remarks on the use of Maude as a multiformalism framework

It can be posited that the integration of multiformalism modeling within a unified framework such as Maude may fundamentally contradict the inherent principles of the multiformalism paradigm, thus reducing its inherent advantages. However, Maude has been employed as a logical framework that accommodates a variety of formalisms and models, ranging from distinct classes of PN to CCS, CSP, BPM, UML dynamic diagrams and queue networks, among others, while preserving their unique characteristics. Our objective is to utilize Maude as a formal ecosystem where these diverse models interact through a specific protocol (a kind of distributed shared memory), operating under semantics grounded in rewriting.

It is pertinent to highlight three significant aspects: within the case study, the shared state is constituted by a multiset of places, synonymous with integer variables. However, the theory underlying the state definition permits alternative representations, such as the implementation of continuous state variables as done in hybrid Petri nets. Each node within the heterogeneous network may possess an internal (or private) state segment, which is subject to transformation via appropriate local rewrites that can be effectively encoded utilizing Maude. Finally, the inherent modular design supports the potential for scaling complexity by allowing individual analysis and subsequent replacement (through rewriting) of network components, or sub-models, with suitable stochastic approximations. For example, under specific circumstances, PT transitions might be combined, or an MQN could be simplified to a basic queue structure.

6 Experimental evidence

Table 1 presents some experimental results related to the Maude encoding of the case study. They refer to the transition system (TS) generated by the parametric alias `netsys(N)` for increasing values of N (the initial population of the network). For example, the following command – which has no solutions for any N– searches for final states throughout the TS. Using slightly different shapes of the command, we can check other base properties, e.g. preservation of the population in the system. Note that actual performance measurement (e.g., system response time) has not been reported since it depends only on the parameters of the model, and their computation time is independent of the actual values provided. As an illustrative example, we report system throughput in the last column (assuming that all structural reconfigurations have a rate of 0.05; we

solved the ergodic CTMC isomorphic to TS, derived using the approach defined in [6]). Instead, the table shows the relative scalability of the proposed approach, highlighting that relatively large systems can be analyzed in a few dozens or hundreds of seconds on a conventional laptop equipped with an 11th-Gen Core i5 and 32 GB RAM. Noticeably, solution 2 shows better performance as N grows, likely due to the different encoding of inner node state transitions (more frequent than rewrites due to structural changes).

```
Maude> search in MQN-SPN : netsys(N) =>! F:NetSys .
```

Table 1. Transition System build of the case-study

N	# states	build time Sol 1 (sec)	build time Sol 2 (sec)	Thr (jobs/sec)
10	5.148	2	2	2,3
20	31.878	19	18	4,1
30	98.208	33	31	6,1
40	222.138	43	38	10,7
50	421.668	56	47	14,4
60	714.798	73	68	19,8
70	1.119.528	318	240	24,8
80	1.653.858	1.480	1.162	29,7
90	2.335.778	3.429	2.924	35,5
100	3.183.318	5.290	4.065	39,2

7 Conclusions

We have presented a multiformalism modeling framework for reconfigurable or adaptive distributed systems integrally realized using the `Maude` system. The case study discussed illustrates the practical benefits of this approach, showing improved modeling flexibility, modularity, and computational efficiency.

Ongoing research is proceeding along two primary trajectories: In the short to medium term, our objective is to incorporate the `Maude` rewrite engine into the SIMTHESys multiformalism framework, which should substantially augment the modeling capacities of the framework. Within SIMTHESys, `Maude` should serve as a solution engine operating on the structural aspects of the model during the solution phase, thus facilitating the dynamic reconfiguration of the model at solution time. This capability would allow for the introduction of new features without necessitating a redesign of either the conceptual framework or the SIMTHESys*ER* solvers generation tool.

Simultaneously, we aim to extend the `Maude` multiformalism framework by integrating compositional operators that highlight network symmetries (automorphisms in colored "hierarchical" graphs) with the objective of deriving a quotient transition system, which is isomorphic to a lumped Markov process, similar to the methodology presented in [8] for rewritable SPN.

References

1. Barbierato, E., Gribaudo, M., Iacono, M.: Modeling hybrid systems in SIMTHESys. Electronic Notes in Theoretical Computer Science **327**, 5 – 25 (2016). https://doi.org/10.1016/j.entcs.2016.09.021
2. Barbierato, E., Gribaudo, M., Iacono, M., Jakóbik, A.: Exploiting CloudSim in a multiformalism modeling approach for cloud based systems. Simulation Modelling Practice and Theory **93**, 133 – 147 (2019). https://doi.org/10.1016/j.simpat.2018.09.018
3. Bause, F., Buchholz, P., Kemper, P.: A toolbox for functional and quantitative analysis of deds. In: Proceedings of the 10th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools. pp. 356–359. TOOLS '98, Springer-Verlag, London, UK (1998)
4. Bruni, R., Meseguer, J.: Generalized rewrite theories. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) Automata, Languages and Programming. pp. 252–266. Springer-Verlag, Berlin, Heidelberg (2003). https://doi.org/10.1007/3-540-45061-0_22
5. Capra, L.: Rewriting logic and Petri nets: A natural model for reconfigurable distributed systems. In: Bapi, R., Kulkarni, S., Mohalik, S., Peri, S. (eds.) Distributed Computing and Intelligent Technology. pp. 140–156. Springer International Pub., Cham (2022). https://doi.org/10.1007/978-3-030-94876-4_9
6. Capra, L.: Associating a Markov process with Maude executable modules. In: Proceedings of the 15th International Conference on Simulation and Modeling Methodologies, Technologies and Applications. pp. 106–116. SciTePress (2025)
7. Capra, L., Gribaudo, M.: A lumped CTMC for modular rewritable PN. In: Doncel, J., Remke, A., Pompeo, D.D. (eds.) Computer Performance Engineering - 20th European Workshop, EPEW 2024, Venice, Italy, June 14, 2024, Revised Selected Papers. Lecture Notes in Computer Science, vol. 15454, pp. 106–120. Springer (2024). https://doi.org/10.1007/978-3-031-80932-3_8
8. Capra, L., Köhler-Bußmeier, M.: Modular rewritable Petri nets: An efficient model for dynamic distributed systems. Theoretical Computer Science **990**, 114397 (2024). https://doi.org/https://doi.org/10.1016/j.tcs.2024.114397
9. Ciardo, G., Miner, A.S.: SMART: The stochastic model checking analyzer for reliability and timing. Quantitative Evaluation of Systems, International Conference on **0**, 338–339 (2004)
10. Clavel, M., Durán, F., Eker, S., Lincoln, P., Oliet, N.M., Meseguer, J., Talcott, C.: All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic. Lecture Notes in Computer Science, Springer (July 2007). https://doi.org/10.1007/978-3-540-71999-1
11. Deavours, D.D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J.M., Sanders, W.H., Webster, P.G.: The Möbius framework and its implementation (2002)
12. Franceschinis, G., Gribaudo, M., Iacono, M., Mazzocca, N., Vittorini, V.: Drawnet++: Model objects to support performance analysis and simulation of systems. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **2324 LNCS**, 233 – 238 (2002). https://doi.org/10.1007/3-540-46029-2_18
13. Gribaudo, M., Iacono, M.: Theory and application of multi-formalism modeling (2013). https://doi.org/10.4018/978-1-4666-4659-9

14. Iacono, M., Gribaudo, M.: Element based semantics in multi formalism performance models. p. 413 – 416 (2010). https://doi.org/10.1109/MASCOTS.2010.54
15. de Lara, J., Vangheluwe, H.: Atom3: A tool for multi-formalism and metamodelling. In: Kutsche, R.D., Weber, H. (eds.) FASE. Lecture Notes in Computer Science, vol. 2306, pp. 174–188. Springer (2002)
16. Sanders, W.: Integrated frameworks for multi-level and multi-formalism modeling. In: Petri Nets and Performance Models, 1999. Proceedings. The 8th International Workshop on. pp. 2–9 (1999). https://doi.org/10.1109/PNPM.1999.796527
17. Trivedi, K.S.: Sharpe 2002: Symbolic hierarchical automated reliability and performance evaluator. In: DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks. p. 544. IEEE Computer Society, Washington, DC, USA (2002)

A HASKELL encoding for reconfigurable timed systems

Antonio César Castro Iglesias[1][0000−0002−8531−6535], Alexandre Madeira[1,2][0000−0002−0646−2017], and Manuel Martins[1,2][0000−0002−5109−8066]

[1] University of Aveiro
[2] CIDMA - Research Center in Mathematics and Applications

Abstract. We introduce a HASKELL implementation of *Reconfigurable Timed Automata* (ReTA), providing an interactive interface that guides users in constructing such models by specifying locations, invariants, transitions, guards, resets, hyper-edges and also allowing for the construction of composed ReTA that can operate together via shared-actions. The tool automatically generates the initial configuration of the defined systems, simulates discrete and delay actions, checks invariance and guard conditions, updates clocks and executes the activating/deactivating effects of the hyper-edges. Also, this encoding allows the users to define traces that record both successful and failed actions, supporting detailed behavioral analysis. As a case study, we introduce a supply chain mechanism, that can be managed by one or two workers, allowing for the definition of composed systems that interact between each other.

Keywords: Reconfigurable Timed Automata · HASKELL · System modeling.

1 Introduction

The increasingly complex behavior of time-dependent systems push the limits of traditional state-based modeling. For that, ReTA automata try to extend the classical notions of such timed systems [1] with the capabilities of reconfigurable systems [3]. For the analysis of such systems, the need of suitable tools emerge, as well known modal checkers such as Uppaal [2] or Romeo [4] are not yet suited with the capabilities for a direct analysis of these novel structures. In this work we present a HASKELL [5] encoding of ReTA. Our implementation guides the user through an incremental construction of ReTA models, generating configurations, simulating both discrete and delay actions and managing hyper-edge timers for dynamic activation and deactivation of transitions. This interactive environment not only makes ReTA accessible for experimentation but also lays the groundwork for future integration with temporal modal logics and other model checking formalisms. As the cornerstone of this encoding, we first present the formal definition of ReTA.

Definition 1. *A* multi-actions Timed Reconfigurable Automata - (ReTA) *is a tuple* $M = (S, X, Act, E, \twoheadrightarrow_T, \nrightarrow_T, Inv, R, \alpha_0, s_0)$ *where:*

- S *is the* set of locations.
- Act *is* the set of actions.
- X *is* the set of clocks.
- $E \subseteq S \times CC(X) \times Act \times 2^X \times S$ *is the set of* ground edges.
- $\rightarrowtail_T \subseteq H(E)$ *is the set of* deactivation edges *and* $\twoheadrightarrow_T \subseteq H(E)$ *is the set of* activating edges. *The set of* hyper-level edges *is given by the disjoint union* $H = \twoheadrightarrow_T \uplus \rightarrowtail_T$
- $Inv : S \to CC(X)$ *is the* invariant assignment function to states
- $R : H \to \mathbb{N}_{\geq 0}$ *is the* times assignment function to hyper-edges
- *the* initial active timed transitions $\alpha_0 = (\alpha_0^E, \alpha_0^{\twoheadrightarrow}, \alpha_0^{\rightarrowtail})$, *where* $\alpha_0^E \subseteq E$, $\alpha_0^{\twoheadrightarrow} \subseteq \twoheadrightarrow$ *and* $\alpha_0^{\rightarrowtail} \subseteq \rightarrowtail$ *are the sets of initial active* ground edges, activation and deactivation arrows*, respectively; and* s_0 *is the* initial location of the system.

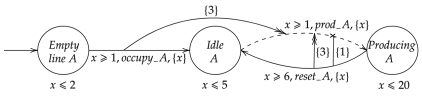
The intuition behind this structure is based on the homogenization of the formalisms of timed automata together with the reconfigurable elements of reactive graphs, where an additional layer of complexity is introduced by incorporating the concept of *time-consuming actions* or *actions that take time*, implemented as timers on the hyper-edge transitions as portrayed in Fig.1. These timers begin their countdown once their associated ground edge is traversed, creating a temporal dependency in the new reconfiguration of our timed reconfigurable automaton, since, once the timer duration has elapsed, a reconfiguration effect will be triggered through the activation or deactivation of the target edge of the hyperedge whose countdown has reached zero.

2 Case Study

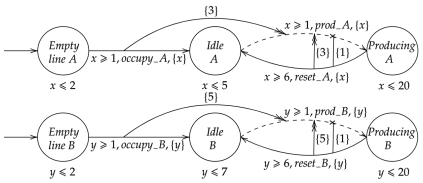
We present a case study on assembly lines of a factory to illustrate the functionalities of ReTA transitions and their reproducibility in HASKELL. Assembly lines create dependencies among products and production stages, activating or deactivating lines as requirements are met. We define a single line to show the temporal and reactive process, later extended into interleaved cases, where one operator manages two lines or two operators run them concurrently. Finally, we show intrusive processes, where one line remains inactive until triggered by the completion of another.

Single line process: Consider now the process portrayed in Fig.1a. Consider the clock x, that will regulate the temporal behavior of the system. The production line operates as follows. The process begins with an empty machine (*Empty line A* location) that requires a worker to become active (*occupy_A* action). As expected, production cannot start without an operator. Once the machine is occupied, the worker takes some time to activate it (*Idle_A* location). After activation (*prod_A* action), the machine enters the production phase (*Producing_A* location), which continues for a limited duration defined by the location invariant, that represents the machine maximum operating capacity. When the system transitions back to the idle state (*reset_A* action), the machine must undergo a cool-down period. During this time, production is temporarily halted.

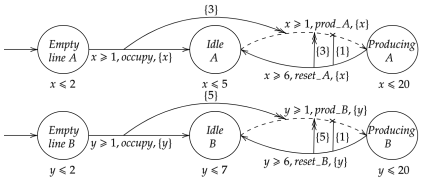
Only after the cool-down time has elapsed can the machine be reactivated and resume its operation.



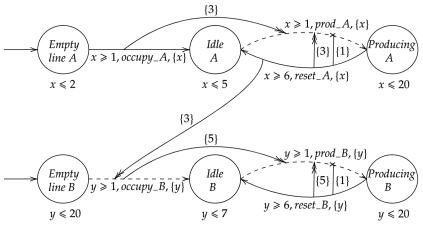
(a) Single machine definition



(b) Supply chain with two workers



(c) Supply chain with one worker



(d) Intrusive communication between machines

Fig. 1: Supply chain configurations: (a) single machine, (b) two workers, (c) one worker, and (d) intrusive communication.

The definition of ReTA in HASKELL is carried out in five phases where the program will prompt the user with a series of questions to define all the components of the system.

1. Define if the system is composed by one or more ReTA.
2. Define system clocks by specifying their names.
3. Define the number of locations, the names of those locations, the clock conditions, their comparison operators ($\leq$, $\geq$,...), and the integer values for the comparisons.
4. Define the system edges similarly to the locations. Specify the source and destination locations for each edge, assign the time guard, indicate whether the edge is active or inactive, and provide the name of the action that triggers the transition.
5. Define the hyper-edges by specifying the source ReTA and the target ReTA. Note that hyper-edges within the same ReTA are referenced as both the source and target. Define the hyper-edges names and an associated timer.

To show the functionalities of the HASKELL encoding, we first consider the system in Fig.1a and we will reproduce its possible behaviors using our code. Finally we also present the final trace of action 2b for our system after a series of alternating discrete and delay actions. Note that this encoding also has the capacity of detecting possible fails of the system by means of invariant violation, impossible transitions due to guard violations and non-active transitions.

Double line process: In this subsection we introduce an exemplification of a supply chain with two production lines. Fig.1b, Fig.1c and Fig.1d portray two process, one operated with two workers (parallel) and the other one operated with just one worker (hand-shaking). Finally we illustrate in Fig.1d the intrusive process, where one of the lines remains inactive until the previous one completes a certain process. In the new example, chain B operates like chain A: a worker activates the machine and starts production. Different invariants and hyper-edge durations reflect machine-specific activation, cooling, and reactivation times. To begin, a delay of 1–2 time units ensures guards on x and y are met and invariants are respected. The system can then move to either $Idle_A$ or $Idle_B$. Afterward, product A can start after 3 time units, while B starts after 5–7, depending on timing constraints. The next step illustrates the behavior of the dual-chain system when both machines are operated by a single worker. In this case, we introduce a shared action *occupy*, representing that both machines are managed by the same operator. Unlike the independent case, this shared action is synchronized and causes a simultaneous transition in the locations of both chains A and B. Fig. 3 shows the shared-actions process on the first discrete transition of the system, where the *occupy* action moves both ReTA to the $Idle_A$ and $Idle_B$ locations. To further illustrate the system 1 evolution, Fig. 3 provides a trace of mixed actions (delay, shared-actions, and parallel).

Intrusive communication: Intrusive communication extends this model by creating dependencies between production lines. In this case, production in line B cannot begin until certain conditions in line A are satisfied. Specifically, the initial transition $occupy_B$ remains inactive until line A completes production or supplies enough output. Once this occurs, an intrusive transition from A activates B, effectively "calling" a new operator to start its production, while A continues operating normally. This mechanism can be scaled to multiple lines, supporting both sequential and parallel flows, as shown in Fig. 1d. The HASKELL output in Fig. 4 further illustrates this, where the hyper-edge $HE2$ generated in line A activates $Edge_4$ from line B after a delay of three time units.

3 Conclusions and future work

We aim to extend the theoretical elements of ReTA by defining suitable modal temporal logics, the definition of bisimulation principles and modal checking techniques. To follow such theoretical advancements we seekto extend the encoding with advanced compositional operators and automated property checking, including the integration of simulation and bisimulation analysis. Finally, the implementation of this encoding into MARGE [6] analysis tool is another main task, offering a complete environment for model construction, verification, and performance evaluation of ReTA models. For reproducibility and research purposes the HASKELL model is available at the Gitlab repository `https://gitlab.com/antonio.iglesias13/haskell_reta_encoding`.

References

1. Alur, R., Dill, D.L.: The theory of timed automata. In: de Bakker, J.W., et al. (eds.) Real-Time: Theory in Practice, REX Workshop, Lecture Notes in Computer Science, vol. 600, pp. 45–73. Springer (1991). https://doi.org/10.1007/BFb0031987, https://doi.org/10.1007/BFb0031987
2. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Formal Methods for the Design of Real-Time Systems, pp. 200–236. Springer (2004). https://doi.org/10.1007/978-3-540-30080-9_7, https://doi.org/10.1007/978-3-540-30080-9_7
3. Gabbay, D.M.: Introducing reactive kripke semantics and arc accessibility. In: Avron, A., et al. (eds.) Pillars of Computer Science, pp. 292–341. Springer (2008). https://doi.org/10.1007/978-3-540-78127-1_17, https://doi.org/10.1007/978-3-540-78127-1_17
4. Gardey, G., Lime, D., Magnin, M., Roux, O.: Romeo: A tool for analyzing time petri nets. In: International Conference on Computer Aided Verification. pp. 418–423. Springer, Berlin, Heidelberg (2005). https://doi.org/10.1007/11513988_41, https://doi.org/10.1007/11513988_41
5. Hudak, P., Fasel, J.H.: A gentle introduction to haskell. ACM SIGPLAN Notices **27**(5), 1–52 (1992). https://doi.org/10.1145/130697.130698, https://doi.org/10.1145/130697.130698
6. Tinoco, D., Madeira, A., Martins, M.A., Proença, J.: Reactive graphs in action. In: Marmsoler, D., Sun, M. (eds.) FACS 2024, Lecture Notes in Computer Science, vol. 15189, pp. 97–105. Springer (2024). https://doi.org/10.1007/978-3-031-71261-6_6, https://doi.org/10.1007/978-3-031-71261-6_6

A Appendix (HASKELL runs)

```
--- Mantra M1 ---
  Loc: EmptyA, Clocks: [("x",0)], Omega: [], ActiveEdges: fromList ["Edge1","Edge3"], ActiveHyper: fromList ["HE1"]
----------------------------
```

```
--- Mantra M1 ---
  Loc: IdleA, Clocks: [("x",1)], Omega: [("HE1",2)], ActiveEdges: fromList ["Edge1","Edge3"], ActiveHyper: fromList ["HE1"]
----------------------------
```

```
--- Mantra M1 ---
  Loc: IdleA, Clocks: [("x",5)], Omega: [], ActiveEdges: fromList ["Edge1","Edge2","Edge3"], ActiveHyper: fromList []
----------------------------
```

(a) Initial configuration | After first discrete transition and 1-unit delay | After hyperedge effects

```
--- Composed System State ---
Shared Actions: []
Total Execution Time: 14 time units
Trace:
  Delay Action: 1 time units
  Discrete Action: 'ocA' by Mantra 'M1'
  Delay Action: 1 time units
  [FAIL] Discrete transition failed for action: prodA by mantra: M1. Shared status: False
  Delay Action: 4 time units
  Discrete Action: 'prodA' by Mantra 'M1'
  Delay Action: 5 time units
  [FAIL] Discrete transition failed for action: resA by mantra: M1. Shared status: False
  [FAIL] TIMELOCK PREVENTED: Delay of 21 violates an invariant in at least one mantra.
  Delay Action: 3 time units
  Discrete Action: 'resA' by Mantra 'M1'
```

(b) Trace of actions for system Fig. 1a

Fig. 2: System evolution and action trace. (a) Some possible configurations; (b) Trace of actions.

```
--- Current System State ---

--- Composed System State ---
Shared Actions: ["oc"]
Total Execution Time: 2 time units
Trace:
  Delay Action: 2 time units
  Shared Discrete Action: 'oc'


--- Mantra M2 ---
  Loc: IdleB, Clocks: [("y",0)], Omega: [("HE2",5)], ActiveEdges: fromList ["Edge4","Edge6"], ActiveHyper: fromList ["HE2"]
--- Mantra M1 ---
  Loc: IdleA, Clocks: [("x",0)], Omega: [("HE1",3)], ActiveEdges: fromList ["Edge1","Edge3"], ActiveHyper: fromList ["HE1"]
----------------------------
```

```
--- Composed System State ---
Shared Actions: ["oc"]
Total Execution Time: 8 time units
Trace:
  Delay Action: 2 time units
  Shared Discrete Action: 'oc'
  Delay Action: 3 time units
  Discrete Action: 'prodA' by Mantra 'M1'
  [FAIL] Discrete transition failed for action: prodB by mantra: M2. Shared status: False
  Delay Action: 3 time units
  Discrete Action: 'prodB' by Mantra 'M2'


--- Mantra M2 ---
  Loc: ProdB, Clocks: [("y",0)], Omega: [], ActiveEdges: fromList ["Edge4","Edge5","Edge6"], ActiveHyper: fromList []
--- Mantra M1 ---
  Loc: ProdA, Clocks: [("x",3)], Omega: [], ActiveEdges: fromList ["Edge1","Edge2","Edge3"], ActiveHyper: fromList []
----------------------------
```

Fig. 3: Shared-actions trace and parallel run trace

```
--- Current System State ---

--- Composed System State ---
Shared Actions: []

--- Mantra M2 ---
  Loc: EmptyB, Clocks: [("y",11)], Omega: [], ActiveEdges: fromList ["Edge6"], ActiveHyper: fromList ["HE2","HE3"]
--- Mantra M1 ---
  Loc: IdleA, Clocks: [("x",0)], Omega: [("HE2",3)], ActiveEdges: fromList ["Edge1","Edge2","Edge3"], ActiveHyper: fromList ["HE2"]
----------------------------
```

```
--- Current System State ---

--- Composed System State ---
Shared Actions: []

--- Mantra M2 ---
  Loc: EmptyB, Clocks: [("y",14)], Omega: [], ActiveEdges: fromList ["Edge4","Edge6"], ActiveHyper: fromList ["HE2","HE3"]
--- Mantra M1 ---
  Loc: IdleA, Clocks: [("x",3)], Omega: [], ActiveEdges: fromList ["Edge1","Edge2","Edge3"], ActiveHyper: fromList []
----------------------------
```

Fig. 4: Intrusive effects in HASKELL

Reasoning about blurred observations of program states: A recipe

Manisha Jain[1,2], Alexandre Madeira[2], and Luís S. Barbosa[3]

[1] INL, Portugal
[2] CIDMA, Dep. Mathematics, Aveiro University, Portugal
[3] HasLab INESC TEC, Dep. Informatics, Minho University, Portugal

Abstract. This short paper presents a simple method to handle fuzzy perceptions, or observations, in the analysis of transition systems and how a modal logic should be fuzzified to handle fuzzy perception. The 'recipe' is illustrated by its application to two cases, The second one, involving quantum programs, may be particularly relevant for the near future.

1 Introduction

What if, when reasoning about a program, the perception of what is the current state is somehow blurred? I.e., what if one cannot assume to be in a definite execution point, but resort instead to a probability distribution or to some sort of weighted vector of computational states? This may come from different phenomena, for example, limited information provided by sensors, in a hybrid program, lack of deterministic control over large learning sets, or when the notion of a state abstracts from data exchanged through multiple interactions in a distributed setting. In the specific case of quantum computing [9] uncertainty is not an oddity, but a crucial part of the game. Actually, not only the result of measuring a quantum state is always probabilistic, but also decoherence further blurs the way the quantum state may evolve or simply be observed.

Fuzzy sets and their theory [11, 5] provides a classical, standard response to model uncertainty. However, in the engineering of programming an extra mile is required to systematically introduce fuzziness in whatever logic program properties are stated and reasoned about. This short paper proposes a very simple, but effective way to *fuzzify* a particular program logic, starting from what we call a *fuzzy perception* of the underlying state space. This propagates to underlying transition system, and even to the way properties are formulated and verified inside the logic, i.e. to the very notion of logic satisfaction. The whole process, named *fuzzification* in the sequel, is illustrated with a couple of different program logics.

After recalling a few elementary concepts, in particular that of a residuated lattice which provides the *truth space* for *fuzzified* logics, the general method is described in Section 2. The whole approach is then instantiated in Section 3 to a few program logics, namely the standard Dynamic Logic [6], and a quantum variant [2, 1].

2 Fuzzifying a modal logic

Our starting point is to define a truth space for the resulting logic, based on a *residuated lattice* $\mathcal{A} = \langle A, \sqcap, \sqcup, 1, 0, \odot, \rightharpoonup, e\rangle$ over a nonempty set A, i.e. a complete lattice $\langle A, \sqcap, \sqcup\rangle$, equipped with a monoid $\langle A, \odot, e\rangle$ such that $\odot$ has a right adjoint, $\rightharpoonup$, the *residuum*. Requiring that the lattice meet ($\sqcap$) and monoidal composition ($\odot$) coincide, and enforcing to $1 = e$, allows the adjunction to be written as $a \sqcap b \leq c$ iff $b \leq a \rightharpoonup c$. Finally, a pre-linearity condition $((a \rightharpoonup b) \sqcup (b \rightharpoonup a) = 1)$ is enforced, opening a wide spectrum of alternative semantics. Examples include Boolean algebras, and the *Gödel lattice* $\boldsymbol{G} = \langle [0,1], \min, \max, 1, 0, \rightarrow\rangle$, with implication defined as $a \rightarrow b = 1$ (if $a \leq b$) or b otherwise. We can now move to the description of the 'fuzzyfication' procedure.

Fuzzifying models. The starting point is a transition system $\mathcal{T} = (W, R)$ over a set Act of (atomic) actions, where W is a set of states and $R = (R_a \subseteq W \times W)_{a \in \mathrm{Act}}$ a family of accessibility relations that interprets Act. Additionally, given a set of propositions Prop one may also consider a valuation $V : \mathrm{Prop} \times W \to \{0, 1\}$ that assigns propositions to states, and write $w \models p$ to abbreviate $V(p, w) = 1$.

Given a residuated lattice $\mathcal{A}$ we define an $\mathcal{A}$- *observation* of $\mathcal{T}$ as $F(\mathcal{T}) = (F(W), F(R))$ where $F(W) = A^W$ is a set of *fuzzy states*, encoding a fuzzy perception of the system states, and $F(R) = (F(R_a) \subseteq F(W) \times F(W))_{a \in \mathrm{Act}}$ denotes the corresponding *accessibility relation*, where, for each $\rho \subseteq W \times W$,

$$F(\rho) = \{(\sigma, \sigma^{(w,v)}) \mid \sigma \in F(W), (w, v) \in \rho, \sigma(w) \neq 0\} \tag{1}$$

and

$$\sigma^{(w,v)}(u) = \begin{cases} \sigma(u), & u \notin \{w, v\} \\ \sigma(w), & u = v \\ 0, & u = w \text{ and } u \neq v \end{cases} \tag{2}$$

The intuition behind this definition is the following. States are now perceptions of the original states, mapping each of the later to a value in $\mathcal{A}$, and the corresponding accessibility relation leads from a perception σ to another one $\sigma^{(w,v)}$, for each original transition from w to v. What is the value of a state u in this new perception $\sigma^{(w,v)}$? If u is not involved in the original transition, i.e. it is not w nor v, the value remains unchanged (i.e. remains as it was in σ). If, on the other hand, u is the target state in (the original) transition, i.e. $u = v$, its value becomes the value in σ of the source state in that transition, i.e. w. This means that the value of w in σ is propagated to the new perception to become there the value of u. If, as final case, u is the source state (i.e. $u = w$) in the original transition leading to a state different from u, clearly u is no more a state that can be possibly part of the new perception, thus taking in $\sigma^{(w,v)}$ the bottom value in the underlying residuated lattice. Note that the actual behaviour of *fuzzified* models is fully determined by the underlying transition structure (W, R). The introduction of fuzziness impacts only on the perceptions.

To illustrate this construction, consider $\mathcal{T} = (W, R)$ over Act $= \{a\}$, where $W = \{s, w, v\}$ and $R_a = \{(w, s), (w, v), (s, v)\}$. Instantiating the semantics with the Gödel lattice $\boldsymbol{G}$, assume the following 'perception of the current state': $\sigma : \{s, w, v\} \to [0, 1]$ such that $\sigma(s) = 0.4$, $\sigma(w) = 0.3$ and $\sigma(v) = 0$ (in a shorter notation we simply write $\sigma = \{s^{0.4}, w^{0.3}\}$). In this case, from σ, we have three possible transitions $F(R_a) = \{(\sigma, \sigma^{(s,v)}), (\sigma, \sigma^{(w,s)}), (\sigma, \sigma^{(w,v)})\}$, where

$$\sigma^{(s,v)} = \{v^{0.4}, w^{0.3}\}, \qquad \sigma^{(w,s)} = \{s^{0.3}\} \quad \text{and} \quad \sigma^{(w,v)} = \{s^{0.4}, v^{0.3}\}. \tag{3}$$

Fuzzifying satisfaction Consider, now, the following quite standard propositional dynamic logic over a signature (Prop, Act), denoted by Fm, for Prop a set of atomic propositions and Act a set of atomic actions:

$$\varphi ::= \bot \mid p \mid \varphi \wedge \varphi \mid \neg\varphi \mid [\pi]\varphi,\ p \in \text{Prop} \tag{4}$$

$$\pi ::= a \mid \pi; \pi \mid \pi + \pi \mid p?,\ a \in \text{Act} \tag{5}$$

with $\pi; \pi'$ meaning the *sequential composition* of the execution of π by the execution of π'; the *choice operator* $\pi + \pi'$ meaning that the execution can do either π or π', and the *iteration operator* π^* meaning that the program repeats π zero or more times. Finally, we have the *test operator* $p?$ that is a program that pursues the execution if the formula p is true there, or abort, if not.

Hence, for a given semantics $\mathcal{A}$, the $\mathcal{A}$-satisfaction is the function $\models^F$: $F(W) \times \mathsf{Fm} \to \mathcal{A}$, where

- $(\sigma \models^F \bot) = 0$
- $(\sigma \models^F p) = \bigsqcup_{w \in W} (\sigma(w) \sqcap V(w, p))$
- $(\sigma \models^F \varphi \wedge \varphi') = (\sigma \models^F \varphi) \sqcap (\sigma \models^F \varphi')$
- $(\sigma \models^F [\pi]\varphi) = \bigsqcap_{\sigma' \in F(W)} ((\sigma, \sigma') \in F(\hat{R}_\pi) \rightharpoonup (\sigma' \models^F \varphi))$

with $\hat{R}_\pi$, the standard interpretation of programs, recursively defined as follows: $\hat{R}_a = R_a$, $a \in Act$; $\hat{R}_{\pi;\pi'} = \hat{R}_\pi \cdot \hat{R}_{\pi'}$, where $\cdot$ is the relational composition; $\hat{R}_{\pi+\pi'} = \hat{R}_\pi \cup \hat{R}_{\pi'}$; $\hat{R}_{\pi^*} = \bigcup_{k \geq 0} \hat{R}^k_\pi$, where $\hat{R}^0_\pi = id$ and $\hat{R}^{k+1}_\pi = \hat{R}_\pi \cdot \hat{R}^k_\pi$; and $\hat{R}_{p?} = \{(w, w) \mid V(w, p) = 1\}$.

If the perception of what counts as the current state is somehow blurred, as assumed here, the result of evaluating a formula in such a state follows a similar principle. This means that the validity of a proposition evaluated in what is perceived as the current state with a specific degree of certainty is weighted by this same degree. Of course in the absence of uncertainty in the perception of the current state, the evaluation becomes entirely defined and the satisfaction relation becomes de standard one.

This procedure is then extended to any formula built from connectives defined over $\mathcal{A}$, and to the interpretation of modalities where the universal quantification over accessible states is given by the generalization of the infimum of $\mathcal{A}$ to arbitrary sets. Note that fuzziness in the interpretation of a program π comes

from the application of F to its standard interpretation given by $\hat{R}_\pi$. As previously observed, the actual behaviour of a program is crisp; the 'locus' of fuzziness are states themselves, as they are perceived as states only up to a uncertainty degree. For illustration purposes, consider $\sigma = \{s^{0.4}, w^{0.3}\}$, and compute

$$\begin{aligned}(\sigma \models^F [a]\top) &= \bigsqcap_{\sigma' \in F(W)} ((\sigma, \sigma') \in F(R_a) \rightharpoonup (\sigma' \models^F \top)) \\ &= (\sigma^{(s,v)} \models^F \top) \sqcap (\sigma^{(w,s)} \models^F \top) \sqcap (\sigma^{(w,v)} \models^F \top) \\ &= (\bigsqcup_{w \in W} (\sigma^{(s,v)}(w) \sqcap (w \models \top))) \sqcap (\bigsqcup_{w \in W} (\sigma^{(w,s)}(w) \sqcap (w \models \top))) \\ &\quad \sqcap (\bigsqcup_{w \in W} (\sigma^{(w,v)}(w) \sqcap (w \models \top))) = 0.3\end{aligned}$$

3 Two further examples

3.1 $\mathcal{A}$-perceptions in the execution of imperative programs

The same construction can be applied to the analysis of a myriad of programming paradigms. Reasoning about imperative programs assuming some level of uncertainty on states observations, provides a first scenario worth to consider. Note that the source of uncertainty can very simply be malfunctions in real devices [10]. Let us revisit the methods introduced in the previous section over a dynamic logic DL [6]. Recall that states in a model for DL are valuations of variables Var over the natural numbers, i.e. $W = \mathbb{N}^{Var}$. Then, programs are specified in the language (5) with actions taken as assignments and propositions understood as equalities and inequalities over numeric expressions (c.f. [6]).

Therefore, consider $\text{Act} = \{x := exp \mid x \in Var \text{ and } exp \in AExp\}$, with $AExp \ni exp ::= x \mid n \mid exp + exp \mid exp - exp \mid exp * exp$, $x \in Var$ and $n \in \mathbb{Z}$ and $Prop \ni p ::= exp = exp \mid exp < exp \mid \neg p \mid p \wedge p$. Clearly, any standard imperative command (e.g. **if** p **then** π_1 **else** $\pi_2 \equiv (p?;\pi_1) + ((\neg p)?;\pi_2)$ and **while** p **do** $\pi \equiv (p?;\pi)^*;\neg p?)$, can be interpreted in this setting. The interpretation of programs extends the interpretation of atoms $x := exp$ as follows,

$$R_{x:=exp} = \{(w, v) \mid v(x) = \hat{w}(exp), \text{ and } v(y) = w(y) \text{ for } y \in Var \text{ with } x \neq y\}$$

where $\hat{w}$ is the natural extension of v from variables to expressions, recursively defined by: $\hat{w}(x) = w(x), x \in Var$; $\hat{w}(n) = n, k \in \mathbb{Z}$; $\hat{w}(exp + exp') = \hat{w}(exp) + \hat{w}(exp')$; and. $\hat{w}(exp * exp') = \hat{w}(exp) * \hat{w}(exp')$. On the other hand, for the interpretation of propositions in $Prop$, consider the valuation $V : W \times Prop \to \{0, 1\}$ recursively defined as follows: $V(w, exp = exp') = 1$ if $\hat{w}(exp) = \hat{w}(exp')$; $V(w, exp < exp') = 1$ if $\hat{w}(exp) < \hat{w}(exp')$; $V(w, \neg p) = 1$ if $V(w, p) = 0$; and $V(w, p \vee p') = 1$ if $V(w, p) = 1$ or $V(w, p') = 1$.

For illustration purposes, consider $Var = \{x\}$, and denote program states by w_i, $i \geq 0$ if $w_i(x) = i$. Hence, we have $R_{x:=x+1} = \{(w_i, w_{i+1}) \mid i \geq 0,\ w_i(x) = i\}$. Program **while** $x \leq 1$ **do** $x := x + 1$ corresponds to the interpretation of the term $(?(x \leq 1)\,;\, x := x + 1)^*;(?(\neg x \leq 1))$, given by relation

$\hat{R}_{(?(x\leq 1)\,;\,x:=x+1)^*;(?(\neg x\leq 1))} = \hat{R}_{(?(x\leq 1)\,;\,x:=x+1)^*} \cdot \hat{R}_{(?(\neg x\leq 1))}$, where $\hat{R}_{(?(x\leq 1)\,;\,x:=x+1)^*}$ is the suprema of

$$\begin{aligned}
\hat{R}^1_{(?(x\leq 1)\,;\,x:=x+1)} &= \hat{R}_{(?(x\leq 1)\,;\,x:=x+1)} = \{(w_0,w_0),(w_1,w_1)\} \cdot \{(w_0,w_1),(w_1,w_2),\cdots\} \\
&= \{(w_0,w_1),(w_1,w_2)\} \\
\hat{R}^2_{(?(x\leq 1)\,;\,x:=x+1)} &= \hat{R}_{(?(x\leq 1)\,;\,x:=x+1)} \cdot \hat{R}^1_{(?(x\leq 1)\,;\,x:=x+1)} \\
&= \{(w_0,w_1),(w_1,w_2)\} \cdot \{(w_0,w_1),(w_1,w_2)\} = \{(w_0,w_2)\} \\
\hat{R}^3_{(?(x\leq 1)\,;\,x:=x+1)} &= \hat{R}_{(?(x\leq 1)\,;\,x:=x+1)} \cdot \hat{R}^2_{(?(x\leq 1)\,;\,x:=x+1)} \\
&= \{(w_0,w_1),(w_1,w_2)\} \cdot \{(w_0,w_2)\} = \emptyset
\end{aligned}$$

Hence, $\hat{R}_{(?(x\leq 1)\,;\,x:=x+1)^*} = id_W \cup \{(w_0,w_1),(w_1,w_2),(w_0,w_2)\}$ and

$$\begin{aligned}
\hat{R}_{\textbf{while } x\leq 1 \textbf{ do } x:=x+1} &= \hat{R}_{(?(x\leq 1)\,;\,x:=x+1)^*;(?(\neg x\leq 1))} \\
&= (id_W \cup \{(w_0,w_1),(w_1,w_2),(w_0,w_2)\}) \cdot \{(w_2,w_2),(w_3,w_3),\cdots\} \\
&= \{(w_0,w_2),(w_1,w_2),(w_2,w_2),(w_3,w_3),\cdots\}
\end{aligned}$$

Consider now a $\boldsymbol{G}$-perception $\sigma = \{w_0^{0.5}, w_1^{0.2}, w_2^{0.4}, w_3^{0.1}\}$. This entails $F(R_{\textbf{while } x\leq 1 \textbf{ do } x:=x+1}) = \{(\sigma,\sigma^{(w_0,w_2)}),(\sigma,\sigma^{(w_1,w_2)}),(\sigma,\sigma^{(w_2,w_2)})\}$, where $\sigma^{(w_0,w_2)} = \{w_1^{0.2}, w_2^{0.4}, w_3^{0.1}\}$, $\sigma^{(w_1,w_2)} = \{w_0^{0.5}, w_2^{0.2}, w_3^{0.1}\}$ and $\sigma^{(w_2,w_2)} = \{w_0^{0.5}, w_1^{0.2}, w_2^{0.4}, w_3^{0.1}\}$.

Finally, program properties can be analyzed in this setting, for instance,

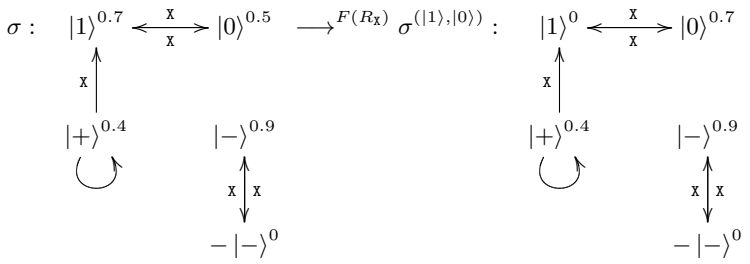
$$\begin{aligned}
&(\sigma \models^F [R_{\textbf{while } x\leq 1 \textbf{ do } x:=x+1}] \models x>1 \\
&= \bigsqcap_{\sigma'\in F(R_{\textbf{while } x\leq 1 \textbf{ do } x:=x+1})} ((\sigma,\sigma')\in F(R_{\textbf{while } x\leq 1 \textbf{ do } x:=x+1}) \rightharpoonup (\sigma' \models^F x>1) \\
&= (1 \rightharpoonup (\sigma^{(w_0,w_2)} \models^F x>1)) \sqcap (1 \rightharpoonup (\sigma^{(w_1,w_2)} \models^F x>1)) \sqcap (1 \rightharpoonup (\sigma^{(w_2,w_2)} \models^F x>1)) \\
&= (\sigma^{(w_0,w_2)} \models^F x>1) \sqcap (\sigma^{(w_1,w_2)} \models^F x>1) \sqcap (\sigma^{(w_2,w_2)} \models^F x>1) \;=\; 0.4
\end{aligned}$$

3.2 $\mathcal{A}$-perceptions on quantum evolutions

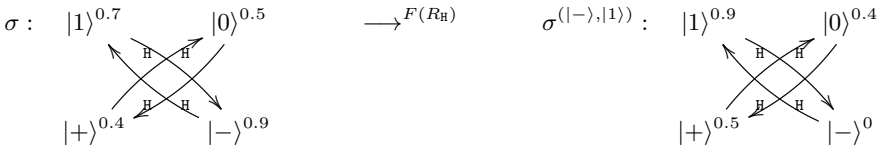
A quantum transition system [2] for a given set of quantum gates names Act, is a pair $\mathcal{T} = (\mathcal{H}, R)$ where, $\mathcal{H}$ is an Hilbert space with states as vectors and $R = (R_{\mathtt{C}} \subseteq \mathcal{H}\times\mathcal{H})_{\mathtt{C}\in \mathrm{Act}}$ is a family of unitary transformations that interprets Act. Additionally, let us take as propositions the state identifiers, say Prop $= \{|0\rangle, |1\rangle, \dots\}$, and consider a valuation function $V : \mathcal{H}\times \mathrm{Prop} \to \{0,1\}$ such that $V(w,|u\rangle) = 1$ if $w = |u\rangle$, mapping propositions into the respective closed Hilbert subspace. Please refer to [8] for an overview on quantum computing and the Dirac notation used in this example.

Again, the recipe can be applied to deal with $\mathcal{A}$-fuzzy perceptions on the evolution of quantum programs. Clearly, each observation state becomes $\sigma : \mathcal{H} \to A$ to represent a fuzzy perception of the current quantum state. To illustrate the method, assume the $\boldsymbol{G}$-perception $\sigma = \{|1\rangle^{0.7}, |0\rangle^{0.5}, |-\rangle^{0.9}, |+\rangle^{0.4}\}$. The effect of executing a $\mathtt{X}$-gate[4] amounts to $F(R_{\mathtt{X}}) = \{(\sigma,\sigma^{(|1\rangle,|0\rangle)}),(\sigma,\sigma^{(|0\rangle,|1\rangle)}),(\sigma,\sigma^{(|1\rangle,|0\rangle)}),(\sigma,\sigma^{(|+\rangle,|+\rangle)}),(\sigma,\sigma^{(|-\rangle,-|-\rangle)}),(\sigma,\sigma^{(-|-\rangle,|-\rangle)})\}$ where, for instance, $\sigma^{(|1\rangle,|0\rangle)} = \{|0\rangle^{0.5}, |+\rangle^{0.4}, |-\rangle^{0.9}\}$. This transition can be graphically represented as:

[4] $\mathtt{X} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$



One may pursue this exercise with other quantum gates. For instance, the application of an Hadamard gate to σ results in $F(R_{\mathtt{H}}) = \{(\sigma, \sigma^{(|1\rangle,|-\rangle)}), (\sigma, \sigma^{(|-\rangle,|1\rangle)}), (\sigma, \sigma^{(|0\rangle,|+\rangle)}), (\sigma, \sigma^{(|+\rangle,|0\rangle)})\}$. An example of the transition $(|-\rangle, |1\rangle)$ can be depicted as follows:



Finally, let us check the following property in this logic:

$$
\begin{aligned}
(\sigma \models^F \langle \mathtt{H} \rangle |1\rangle) &= \bigsqcup_{\sigma' \in F(W)} ((\sigma, \sigma') \in F(R_{\mathtt{H}})) \sqcap (\sigma' \models^F |1\rangle) \\
&= ((\sigma, \sigma^{(|1\rangle,|-\rangle)}) \in F(R_{\mathtt{H}})) \sqcap (\sigma^{(|1\rangle,|-\rangle)} \models^F |1\rangle) \sqcup ((\sigma, \sigma^{(|-\rangle,|1\rangle)}) \in F(R_{\mathtt{H}})) \sqcap (\sigma^{(|-\rangle,|1\rangle)} \models^F |1\rangle) \sqcup \\
&\quad ((\sigma, \sigma^{(|0\rangle,|+\rangle)}) \in F(R_{\mathtt{H}})) \sqcap (\sigma^{(|0\rangle,|+\rangle)} \models^F |1\rangle) \sqcup ((\sigma, \sigma^{(|+\rangle,|0\rangle)}) \in F(R_{\mathtt{H}})) \sqcap (\sigma^{(|+\rangle,|0\rangle)} \models^F |1\rangle) \\
&= (\sigma^{(|1\rangle,|-\rangle)} \models^F |1\rangle) \sqcup (\sigma^{(|-\rangle,|1\rangle)} \models^F |1\rangle) \sqcup (\sigma^{(|0\rangle,|+\rangle)} \models^F |1\rangle) \sqcup (\sigma^{(|+\rangle,|0\rangle)} \models^F |1\rangle) \\
&= \bigsqcup_{w \in W} (\sigma^{(|1\rangle,|-\rangle)}(w) \models |1\rangle) \sqcup \bigsqcup_{w \in W} (\sigma^{(|-\rangle,|1\rangle)}(w) \models |1\rangle) \sqcup \\
&\quad \bigsqcup_{w \in W} (\sigma^{(|0\rangle,|+\rangle)}(w) \models |1\rangle) \sqcup \bigsqcup_{w \in W} (\sigma^{(|+\rangle,|0\rangle)}(w) \models |1\rangle) \\
&= 0 \sqcup 0.9 \sqcup 0.7 \sqcup 0.7 = 0.9
\end{aligned}
$$

4 Concluding

This short paper introduced a recipe' to analyze transition systems from an external perspective, therefore taking into account possible partial knowledge about the current state in which the system is executing. As discussed above, such situations may arise in many application scenarios, including, but not limited to, the analysis of quantum programs.

The versatility of this generic method in handling other formalizations of quantum transition systems (e.g., quantum automata with mixed states [4]), probabilistic modalities (e.g., [3]), and concurrency operators [7]) is part of ongoing work. The construction of a fuzzified modal logic over another base logic, itself fuzzified', is also part of our current area.

References

1. Alexandru Baltag and Sonja Smets. LQP: the dynamic logic of quantum information. *Math. Struct. Comput. Sci.*, 16(3):491–525, 2006.
2. Alexandru Baltag and Sonja Smets. The logic of quantum programs. *CoRR*, abs/2109.06792, 2021.
3. Alexandru Baltag and Sonja Smets. Reasoning about quantum information: An overview of quantum dynamic logic. *Applied Sciences*, 12(9), 2022.
4. Rūsiņš Freivalds, Māris Ozols, and Laura Mančinska. Improved constructions of mixed state quantum automata. *Theoretical Computer Science*, 410(20):1923–1931, 2009. Quantum and Probabilistic Automata.
5. P. Hajek. *The Metamathematics of Fuzzy Logic*. Kluwer, 1998.
6. David Harel, Jerzy Tiuryn, and Dexter Kozen. *Dynamic Logic*. MIT Press, Cambridge, MA, USA, 2000.
7. Manisha Jain, Vitor Fernandes, and Alexandre Madeira. Adding concurrency to quantum dynamic logic. In *International Conference on AI Logic and Applications*, pages 17–31. Springer, 2024.
8. Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge, 2000.
9. Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011.
10. Thomas G. Wong. *Introduction to Classical and Quantum Computing*. Rooted Grove, 2022.
11. L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

Four-valued logics of indicative conditionals

Miguel Muñoz Pérez[0009−0001−3468−8208]

Departamento de Lógica, Historia y Filosofía de la Ciencia,
UNED, Madrid, Spain
mupemiguel99@gmail.com

Abstract. We detail some ways in which the study of three-valued logics of indicative conditionals can be extended by further adding a new truth-value. Our approach heavily relies on twist constructions, which have been already used in the literature in order to provide algebraic semantics in the three-valued case. Here we follow the inverse path: we first specify how these twist constructions, expanded with the new truth-value, induce new logics and then we prove the corresponding twist representation results.

1 Introduction

In this short communication we wish to continue the study on logics of indicative conditionals as presented in [13,10]. Our intention here is to briefly describe some ways of generalizing logics of indicative conditionals, prominently through the use of *twist structures* (see, e.g., [15,7,16]). For the proofs, detailed observations and comments on further extensions of this approach, we redirect the reader to [12].

Background. By *indicative conditionals* we understand the simplest conditional statements from natural speech – assumed, without loss of generality, to be of the form if-then – that are concerned with what could be true. Much has been said regarding the formalization of indicative conditionals (see, e.g. [4,5] for a survey); for now let us simply point out that, in case the antecedent of an indicative conditional is false, the overall statement seems to lack a definite truth-value and that we may speak of a semantic *gap*. Among the possible ways of formalizing such intuition, we can simply add a new truth-value $\mathbf{1/2}$ to the classical ones $\mathbf{0}$ and $\mathbf{1}$ in such a way that $\mathbf{0} \to \psi = \mathbf{1/2}$. It is also common to consider logics in which *non-falsity* is preserved under inferences, that is, logics having $\mathbf{1/2}$ and $\mathbf{1}$ as designated elements. Of course, the task of determining the behavior of $\mathbf{1/2}$ with respect to the usual connectives is left open (intuitions may differ when settling, e.g., the value of $\mathbf{1/2} \to \psi$). But these constraints are enough to determine the class of what we call (propositional) *logics of indicative conditionals*. Here, we wish to extend our previous analysis of four main cases, namely: *De Finetti's logic* DF [8,4,5], *Cooper's logic of ordinary language* OL [3,16], *Farrell's logic* F [6,13] and *Cantwell's logic of conditional negation* CN [1,13].

Preliminaries. Now, the main idea behind the use of twist structures is that of representing the algebraic semantics associated with a target logic (in our case DF, CN and OL) as a special kind of power construction in terms of more well-known structures (see, e.g. [14,15]). Below we introduce these familiar classes of algebras.

A distributive lattice $(L; \wedge, \vee)$ is a *De Morgan lattice* if there is an operation $\neg : L \to L$ such that both $\neg\neg a = a$ and $\neg(a \wedge b) = \neg a \vee \neg b$ hold for every $a, b \in L$. A De Morgan lattice L is called *centered* if there is some element $c \in L$ such that $\neg c = c$. If L has two distinct centers, we say that it is *bi-centered*. In case a De Morgan lattice is bounded, we speak of a *De Morgan algebra*. A *Kleene algebra* is a De Morgan lattice L in which it holds that $\neg a \wedge a \leqslant \neg b \vee b$, for every $a, b \in L$. On the other hand, recall that a *generalized Boolean algebra* $(B; \wedge, \to)$ is the 0-free reduct of a Boolean algebra $(B; \wedge, \to, 0)$.

The three-valued case. Regarding our target three-valued logics, the overall strategy is common through all the cases: first, one identifies some class of algebras that can be associated (in a relevant sense) with the logic considered in each case; then, one defines a twist structure and check that it is an element of the aforementioned class and, finally, one proves a *twist representation* result, that is, that each member of the initial class is isomorphic to some of these twist algebras. Since we will later only deal with DF, CN and F, we redirect the reader interested in OL to [9,16], where this logic has been deeply studied. In the other cases, we will only explicitly present the algebraic counterpart in each case; for the twist representation results, see [12, §2]. For instance, the algebraic models of DF can be seen as centered Kleene algebras (see above) [2]. For CN, one can define, following [13]:

Definition 1. *A* CN-algebra *is an algebra* $A := (A; \wedge, \rightarrow, \neg, \mathbf{1/2})$ *such that the reduct* $(A; \wedge, \neg, \mathbf{1/2})$ *is a centered Kleene lattice (with* $\vee$ *defined, as usual, through De Morgan's law) and the following equations hold: (CN1)* $(x \wedge y) \rightarrow z = x \rightarrow (y \rightarrow z)$*, (CN2)* $\mathbf{1/2} \leq x \rightarrow (y \rightarrow y)$*, (CN3)* $\mathbf{1/2} \leq ((x \rightarrow y) \rightarrow x) \rightarrow x$*, (CN4)* $\Diamond(x \rightarrow y) = \Diamond x \rightarrow \Diamond y$ *and (CN5)* $\neg(x \rightarrow y) = x \rightarrow \neg y$*, where* $\Diamond x := x \wedge \mathbf{1/2}$*.*

One can similarly prove a twist representation result for this case, which has been already studied in [7]. It is also worth noting that one can alternatively define *F-algebras* and obtain a similar result (again, see [12]):

Definition 2. *A* F-algebra $(A; \neg, \wedge, \vee, \rightarrow, \mathbf{1/2})$ *such that* $(A; \neg, \wedge, \vee, \mathbf{1/2})$ *is a centered Kleene algebra satisfying that: (F1)* $x \wedge y = x \wedge (x \rightarrow y)$*, (F2)* $x \rightarrow y \leq (x \wedge y) \vee \mathbf{1/2}$*, (F3)* $(x \wedge y) \rightarrow z = x \rightarrow (y \rightarrow z)$*, (F4)* $\mathbf{1/2} \leq x \rightarrow (y \rightarrow y)$*, (F5)* $\mathbf{1/2} \leq ((x \rightarrow y) \rightarrow x) \rightarrow x$ *and (F6)* $\Diamond(x \rightarrow y) = \Diamond x \rightarrow \Diamond y$*, where* $\Diamond x := x \wedge \mathbf{1/2}$*.*

2 The four-valued case

The overall strategy carried out in the twist semantics ultimately consists in identifying $\mathbf{0}$ with $(0,1)$, $\mathbf{1/2}$ (from now on, $\top$) with $(1,1)$ and $\mathbf{1}$ with $(1,0)$. But the operations introduced in the corresponding twist structures also admit as argument the pair $(0,0)$. Hence, this pair could be seen as corresponding to the new truth-value $\bot$ through the previous identification. In what follows, we let $A_4 := \{\mathbf{0}, \bot, \top, \mathbf{1}\}$. We will only deal explicitly with DF, CN and F (for more details, also regarding the philosophical motivation of these logics, see [12, §3]).

2.1 Adding a semantic gap

De Finetti. The tables for the corresponding operations are the following ones:

	$\neg$
$\mathbf{0}$	$\mathbf{1}$
$\bot$	$\bot$
$\top$	$\top$
$\mathbf{1}$	$\mathbf{0}$

$\rightarrow_{\mathsf{DF}}$	$\mathbf{0}$	$\bot$	$\top$	$\mathbf{1}$
$\mathbf{0}$	$\top$	$\top$	$\top$	$\top$
$\bot$	$\mathbf{0}$	$\bot$	$\mathbf{0}$	$\bot$
$\top$	$\top$	$\top$	$\top$	$\top$
$\mathbf{1}$	$\mathbf{0}$	$\bot$	$\top$	$\mathbf{1}$

$\wedge_{\mathsf{K}}$	$\mathbf{0}$	$\bot$	$\top$	$\mathbf{1}$
$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$	$\mathbf{0}$
$\bot$	$\mathbf{0}$	$\bot$	$\mathbf{0}$	$\bot$
$\top$	$\mathbf{0}$	$\mathbf{0}$	$\top$	$\top$
$\mathbf{1}$	$\mathbf{0}$	$\bot$	$\top$	$\mathbf{1}$

$\vee_{\mathsf{K}}$	$\mathbf{0}$	$\bot$	$\top$	$\mathbf{1}$
$\mathbf{0}$	$\mathbf{0}$	$\bot$	$\top$	$\mathbf{1}$
$\bot$	$\bot$	$\bot$	$\mathbf{1}$	$\mathbf{1}$
$\top$	$\top$	$\mathbf{1}$	$\top$	$\mathbf{1}$
$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$

The implication $\rightarrow_{\mathsf{DF}}$ is computed by the term that defines it in the three-valued case. In this way, we may define the logic induced by DFg induced by the matrix $(\mathbf{DFg}_4, \{\top, \mathbf{1}\})$, where $\mathbf{DFg}_4 := (A_4; \neg, \wedge_{\mathsf{K}}, \vee_{\mathsf{K}}, \rightarrow_{\mathsf{DF}})$. As we have seen before, we can simply work with $\wedge_{\mathsf{K}}, \vee_{\mathsf{K}}$ and $\top$. A reasonable choice for the twist structures in this case is the following:

Definition 3. *Let* $L := (L; \wedge, \vee, 0, 1)$ *be a bounded distributive lattice. The* full DFg-twist algebra over L *consists in an algebra* $L^{\bowtie}$ *with universe* $L \times L$ *and operations*

i. $(x_1, y_1) \wedge (x_2, y_2) := (x_1 \wedge x_2, y_1 \vee y_2)$,
ii. $\neg(x, y) := (y, x)$,
iii. $\top := (1, 1)$ *and* $\bot := (0, 0)$.
A DFg-twist algebra over L *is any subalgebra* $\mathcal{A} \leqslant L^{\bowtie}$ *such that* $\pi_1[\mathcal{A}] = L$.

Definition 4. *A* DFg-algebra $(A; \wedge, \vee, \neg, 0, \bot, \top)$ *is a bi-centered De Morgan lattice* $(A; \wedge, \vee, \neg, \bot, \top)$ *with lower bound* 0 *and where* $\bot \wedge \top = 0$.

Then, clearly, every DFg-twist algebra is a DFg-algebra. Given a DFg-algebra $A := (A; \wedge, \vee, \neg, 0, \bot, \top)$, define the map $\Diamond : A \to A : x \mapsto x \wedge \top$ and the algebra $\Diamond(A) := (\Diamond(A); \wedge, \vee, 0, \top)$, in which the operations are defined as restrictions of the ones from A. Then, the following is clear by the DF case and noting that 0 is the desired lower bound (see [12]):

Lemma 1. *Given a DFg-algebra A, it holds that* $(\Diamond(A); \wedge, \vee, 0, \top)$ *is a bounded distributive lattice.*

Theorem 1 (DFg-twist representation). *Every DFg-algebra A is isomorphic to a DFg-twist algebra over* $\Diamond(A)$*, as witnessed by the map* $\iota \colon A \to \Diamond(A) \times \Diamond(A) : a \mapsto (\Diamond a, \Diamond \neg a)$.

Cantwell. For CN, the new tables are the following:

$\to_{\mathsf{OL}}$	**0**	$\bot$	$\top$	**1**
0	$\top$	$\top$	$\top$	$\top$
$\bot$	$\top$	$\top$	$\top$	$\top$
$\top$	**0**	$\bot$	$\top$	**1**
1	**0**	$\bot$	$\top$	**1**

$\wedge_{\mathsf{K}}$	**0**	$\bot$	$\top$	**1**
0	**0**	**0**	**0**	**0**
$\bot$	**0**	$\bot$	**0**	$\bot$
$\top$	**0**	**0**	$\top$	$\top$
1	**0**	$\bot$	$\top$	**1**

$\vee_{\mathsf{K}}$	**0**	$\bot$	$\top$	**1**
0	**0**	$\bot$	$\top$	**1**
$\bot$	$\bot$	$\bot$	**1**	**1**
$\top$	$\top$	**1**	$\top$	**1**
1	**1**	**1**	**1**	**1**

We define the logic CNg as induced by the matrix $(\mathbf{CNg}_4, \{\top, \mathbf{1}\})$, where $\mathbf{CNg}_4 := (A_4; \neg, \wedge_{\mathsf{K}}, \vee_{\mathsf{K}}, \to_{\mathsf{OL}})$. This logic has already been considered in [7, p. 61]. We define:

Definition 5. *Let B be a Boolean algebra. We define the* full CNg-twist algebra over B *as the algebra* $B^{\bowtie}$ *with universe* $B \times B$ *and operations*

i. $(x_1, y_1) \wedge (x_2, y_2) := (x_1 \wedge x_2, y_1 \vee y_2)$,
ii. $\neg(x, y) := (y, x)$,
iii. $(x_1, y_1) \to (x_2, y_2) := (x_1 \to x_2, x_1 \to y_2)$,
iv. $\top := (1, 1)$ *and* $\bot := (0, 0)$.
As before, one can define $x \vee y := \neg(\neg x \wedge \neg y)$. *A* CNg-twist algebra over B *is any subalgebra* $\mathcal{A} \leqslant B^{\bowtie}$ *such that* $\pi_1[\mathcal{A}] = B$.

Then, it is clear that $\mathbf{CNg_4}$ is isomorphic to the full CNg-twist algebra over the two-element Boolean algebra by the usual identification. Note that, since $\top$ is immediately definable by $\bot$, we can omit $\top$ in the previous definition. It is clear that every CNg-twist algebra belongs to the following class:

Definition 6. *A* CNg-algebra $(A; \wedge, \vee, 0, \neg, \to \bot, \top)$ *is a DFg-algebra* $(A; \wedge, \vee, \neg, 0, \bot, \top)$ *verifying (CN1)-(CN5) from Definition 1.*

Given a CNg-algebra $(A; \wedge, \vee, 0, \to, \neg, \bot, \top)$, define $\Diamond : A \to A : x \mapsto x \wedge \top$ as before. Similarly as in the DFg case, by restricting the operations we can define the algebra $\Diamond(A) := (\Diamond(A); \wedge, \vee, 0, \to, \top)$. Moreover, the CN case allows us to prove, by simply noting that 0 is a lower bound for the generalized Boolean algebra $(\Diamond(A); \wedge, \vee, \to, \top)$ that:

Lemma 2. *Given a CNg-algebra A, it holds that* $\Diamond(A)$ *is a Boolean algebra.*

Theorem 2 (CNg-twist representation). *Every CNg-algebra A is isomorphic to a CNg-twist algebra over* $\Diamond(A)$*, as witnessed by the map* $\iota \colon A \to \Diamond(A) \times \Diamond(A) : a \mapsto (\Diamond a, \Diamond \neg a)$.

Farrell. The following twist structure allows us to compute the table below:

Definition 7. *Consider a Boolean algebra B. We define the* full Fg-twist algebra over B *as the algebra $B^{\bowtie}$ with universe $B \times B$ and operations*

i. $(x_1, y_1) \wedge (x_2, y_2) := (x_1 \wedge x_2, y_1 \vee y_2)$,
ii. $\neg(x, y) := (y, x)$,
iii. $\top := (1, 1)$ *and* $\bot := (0, 0)$,
iv. $(x_1, y_1) \to (x_2, y_2) := (x_1 \to x_2, y_1 \vee y_2)$.

A Fg-twist algebra over B *is any subalgebra $\mathcal{A} \leqslant B^{\bowtie}$ such that $\pi_1[\mathcal{A}] = B$.*

$\to_{\mathsf{F}}$	**0**	$\bot$	$\top$	**1**
0	$\top$	$\top$	$\top$	$\top$
$\bot$	$\top$	**1**	$\top$	**1**
$\top$	**0**	**0**	$\top$	$\top$
1	**0**	$\bot$	$\top$	**1**

and then define the logic Fg as induced by the matrix $(\mathbf{Fg}_4, \{\top, \mathbf{1}\})$, where $\mathbf{Fg_4} := (A_4; \neg, \wedge_{\mathsf{K}}, \vee_{\mathsf{K}}, \to_{\mathsf{F}})$. Obviously, the preceding remarks on the inter-definability of $\to_{\mathsf{OL}}$ and $\to_{\mathsf{F}}$ still holds, so we can simply refer to the CNg case. However, for the sake of completeness, let us point out that one can define:

Definition 8. *A* Fg-algebra $(A; \wedge, \vee, 0, \neg, \to \bot, \top)$ *is a DFg-algebra* $(A; \wedge, \vee, \neg, 0, \bot, \top)$ *verifying (F1)-(F6) from Definition 2.*

So that every Fg-twist algebra is in fact a Fg-algebra. Moreover, given a Fg-algebra $(A; \wedge, \vee, 0, \to, \neg, \bot, \top)$, we can define $\Diamond : A \to A : x \mapsto x \wedge \top$ as before. Similarly as in the CNg case, by restricting the operations we can define the algebra $\Diamond(A) := (\Diamond(A); \wedge, \vee, 0, \to, \top)$. We already know that $\Diamond(A)$ has a reduct that is a bounded distributive lattice. Moreover, the F case allows us to prove, by simply noting that 0 is a lower bound for the generalized Boolean algebra $(\Diamond(A); \wedge, \vee, \to, \top)$ that:

Lemma 3. *Given a Fg-algebra A, it holds that $\Diamond(A)$ is a Boolean algebra.*

Theorem 3 (Fg-twist representation). *Every Fg-algebra A is isomorphic to a Fg-twist algebra over $\Diamond(A)$, as witnessed by the map $\iota \colon A \to \Diamond(A) \times \Diamond(A) : a \mapsto (\Diamond a, \Diamond \neg a)$.*

2.2 Adding a falsity

De Finetti (kinda). In the case of CNg, working first with DFg was quite instructive, since it allowed use to first study the implication-free fragment of our full language. Here, it seems reasonable to proceed similarly. However, the change of the negation operator compromises the definability results above and, in particular, the definability of $\to_{\mathsf{DF}}$. We will later return to this. For now, we may define DFf as the logic induced by $(\mathbf{DFf}_4, \{\top, \mathbf{1}\})$, where $\mathbf{DFf}_4 := (A_4; -, \wedge_{\mathsf{K}}, \vee_{\mathsf{K}}, \top)$[1] and where $-$ interchanges **0** with **1** and $\bot$ with $\top$. As we announced before regarding twist structures, an inner negation is needed from each factor algebra. Hence, it seems natural to define:

[1] Philosophically, it is clear that De Finetti's logic is *not* captured by having the strong Kleene operations and adding a negation: this seems more like an accidental feature that holds in the DF and DFg cases. Rather, one should consider the full modified language $-, \wedge_{\mathsf{K}}, \vee_{\mathsf{K}}, \to_{\mathsf{DF}}$. However, we are interested in this variation of DF because in provides some background for the next case studies. One could also provide some philosophical vindication of this setting (see below).

Definition 9. *Let D_1, D_2 be two De Morgan algebras. The* full DFf-twist algebra over D_1 and D_2 *consists in an algebra $D_1 \bowtie D_2$ with universe $D_1 \times D_2$ and operations*

i. $(x_1, y_1) \wedge (x_2, y_2) := (x_1 \wedge_{D_1} x_2, y_1 \vee_{D_2} y_2)$,
ii. $-(x, y) := (\neg_{D_1} x, \neg_{D_2} y)$,
iii. $\bot := (0_{D_1}, 0_{D_2})$.

Note that we may simply let $\top := -\bot$ and, similarly, we can define $x \vee y := -(-x \wedge -y)$ and $x \to y := (-x \wedge \top) \vee (x \wedge y)$. A DFf-twist algebra over D_1 and D_2 *is any subalgebra $\mathcal{A} \leqslant D_1 \bowtie D_2$ such that $\pi_1[\mathcal{A}] = D_1$.*

Note how $\mathbf{DFf_4}$ is isomorphic to the DFf-twist algebra in which the factor structures are both the two-element Boolean algebra. Having two different factor algebras in the previous definition is motivated by the behavior of the negation operation. One can check that every DFf-twist algebra is of the following kind:

Definition 10. *A* DFf-algebra $(D; \wedge, \vee, \neg, 0, 1, \top)$ *is a bounded De Morgan lattice $(D; \wedge, \vee, \neg, 0, 1)$ in which it holds that: (DFf1) $\top$ (and $\neg\top$) receive different values to those of 0 and 1*[2]*, (DFf2) $\top \wedge \neg\top = 0$ and (DFf3) $a \wedge \top = b \wedge \top$ and $a \vee \neg\top = b \vee \neg\top$ imply that $a = b$. Observe that we also have $\neg 0 = 1$. We set $\bot := \neg\top$.*

Now, given a DFf-algebra $(D; \wedge, \vee, \neg, 0, 1, \bot, \top)$, set $\Diamond x := x \wedge \top$ and $\Box x := x \wedge \bot$. By restricting the operations from D, one can define $(\Diamond(D); \wedge, \vee, \neg_\Diamond, 0, \top)$ and $(\Box(D); \wedge, \vee, \neg_\Box, 0, \bot)$, where $\neg_\Diamond \Diamond a := \Diamond\neg a$ and $\neg_\Box \Box a := \Box\neg a$. Then, it is clear that these algebras are De Morgan algebras: for instance, one can easily check that both $\neg_\Diamond$ and $\neg_\Box$ preserve tops and bottoms. Then, we can prove:

Theorem 4 (DFf-twist representation). *Let D be a DFf-algebra. Then, the map $\iota : D \to \Diamond(D) \times \Box(D) : a \mapsto (\Diamond a, \Box\neg a)$ witnesses an isomorphism between D and the corresponding DFf-twist structure.*

Cantwell. We define the logic CNf as induced by $(\mathbf{CNf}_4, \{\top, \mathbf{1}\})$, where one sets $\mathbf{CNf}_4 := (A_4; -, \wedge_\mathsf{K}, \vee_\mathsf{K}, \to_{\mathsf{OL}})$. The preceding case of DFf has shown how we need to allow different factor algebras in our twist representation. Not only this but, comparing with Definitions 5, we need some map for the definition of the implication operation, since it is the only one not determined component-wise by the inner operations of the factor algebras:

Definition 11. *Let B_1, B_2 be two Boolean algebras and $\rho : B_1 \to B_2$ an embedding. We define the* full CNf-twist algebra over B_1 and B_2 *as the algebra $B_1 \bowtie B_2$ with universe $B_1 \times B_2$ and operations*

i. $(x_1, y_1) \wedge (x_2, y_2) := (x_1 \wedge_{B_1} x_2, y_1 \vee_{B_2} y_2)$,
ii. $-(x, y) := (\neg_{B_1} x, \neg_{B_2} y)$,
iii. $(x_1, y_1) \to (x_2, y_2) := (x_1 \to_{B_1} x_2, \rho(x_1) \to_{B_2} y_2)$,
iv. $\bot := (0_{B_1}, 0_{B_2})$.

Additionally, one can define $x \vee y := -(-x \wedge -y)$. A CNf-twist algebra over B_1 and B_2 *is any subalgebra $\mathcal{A} \leqslant B_1 \bowtie B_2$ verifying that $\pi_1[\mathcal{A}] = B_1$.*

Note that in this case we have that $\top$ is definable by $-x \to (x \to x)$ and hence $\bot$ is too by letting $\bot := -\top$ (alternatively, we may set $\top := (0, 1) \to (1, 0)$ and

[2] As a quasi-equation, we can see this condition as $\top \approx 1 \Rightarrow x \approx y$, so that only the trivial algebra can identify $\top$ and 1 (and their negations).

$\bot := -\top$)[3]. Again, as expected, $\mathbf{CNf_4}$ is isomorphic to the CNf-twist algebra over two copies of the two-element Boolean algebra (where ρ is the identity)[4].

Farrell. The virtue of Farrell's implication in twist structures is that, unlike Cooper's, it can be defined component-wise. Since one of the factors simply needs less operations, we can define this twist structure from mixed factor algebras:

Definition 12. *Let B and D be a Boolean algebra and a De Morgan algebra, respectively. We define the* full Ff-twist algebra over B and D *as the algebra $(B \times D)^{\bowtie}$ with universe $B \times D$ and operations*

i. $(x_1, y_1) \wedge (x_2, y_2) := (x_1 \wedge_B x_2, y_1 \vee_D y_2)$,
ii. $-(x, y) := (\neg_B x, \neg_D y)$,
iii. $(x_1, y_1) \to (x_2, y_2) := (x_1 \to_B x_2, y_1 \vee_D y_2)$,
iv. $\bot := (0_B, 0_D)$.

Additionally, one can set $x \vee y := -(-x \wedge -y)$. A Ff-twist algebra over B and D *is any subalgebra $\mathcal{A} \leqslant (B \times D)^{\bowtie}$ such that $\pi_1[\mathcal{A}] = B$.*

We can define $\mathbf{Ff_4} := (A_4; -, \wedge_{\mathsf{K}}, \vee_{\mathsf{K}}, \to_{\mathsf{F}})$ and take the logic Ff to be induced by the matrix $(\mathbf{Ff_4}, \{\top, \mathbf{1}\})$. The reader may wonder whether we could restrict to this case in order to study CNf. However, as we pointed out regarding DFf, the change in the negation operation has several implications, namely, that $\to_{\mathsf{OL}}$ fails to be definable by means of $\to_{\mathsf{F}}$. Moreover, it turns out that the term $(\top \wedge \neg x) \vee (x \wedge y)$ defines $\to_{\mathsf{F}}$ so that, *prima facie*, we can restrict ourselves to the isomorphism of Theorem 4. Now, turning to the twist representation result, let us first define an appropriate class of algebras:

Definition 13. *A* Ff-algebra $(D; \wedge, \vee, \neg, \to, 0, 1, \bot, \top)$ *is a DFf-algebra $(D; \wedge, \vee, \neg, 0, 1, \bot, \top)$ in which the following conditions hold: (Ff1) $x \wedge y = x \wedge (x \to y)$, (Ff2) $(x \wedge y) \to z = x \to (y \to z)$, (Ff3) $\top \leq ((x \to y) \to x) \to x$, (Ff4) $\Diamond(x \to y) = \Diamond x \to \Diamond y$ and (Ff5) $\Box\neg(x \to y) = \Box\neg(x \wedge y)$, where $\Diamond x := x \wedge \top$ and $\Box x := x \wedge \bot$.*

One can check that every Ff-twist algebra is in fact a Ff-algebra. Now, define similarly as in the DFf case: $(\Diamond(D); \wedge, \vee, \neg_\Diamond, \to, 0, \top)$ and $(\Box(D); \wedge, \vee, \neg_\Box, 0, \bot)$, where, remember, $\neg_\Diamond \Diamond a := \Diamond\neg a$ and $\neg_\Box \Box a := \Box\neg a$.

Lemma 4. *Given a Ff-algebra $(D; \wedge, \vee, \neg, \to, 0, 1, \bot, \top)$, it holds that $\Diamond(D)$ is a Boolean algebra and that $\Box(D)$ is a De Morgan algebra.*

Theorem 5 (DFf-twist representation). *Let D be a Ff-algebra. Then, the map $\iota : D \to \Diamond(D) \times \Box(D) : a \mapsto (\Diamond a, \Box\neg a)$ witnesses an isomorphism between D and the corresponding Ff-twist structure.*

Acknowledgments. I would like to thank Umberto Rivieccio for his useful comments and for providing me with some crucial insights, as well as the anonymous referees for their suggestions on how to improve the present paper. I hope to have properly incorporated them into the text.

[3] Another observation is that, having all four constants available, from the 'truth order' $\wedge, \vee$ one can explicitly define, via the so-called *90-degree lemma* [11, Lemma 1.5], the 'knowledge order', usually considered on bi-lattices (see, e.g., [11]). The definitions are: $x \sqcap y := (x \wedge \bot) \vee (y \wedge \bot) \vee (x \wedge y)$ and $x \sqcup y := (x \wedge \top) \vee (y \wedge \top) \vee (x \wedge y)$. Additionally, one can compute their point-wise form in twist algebras.

[4] Note that the new negation prevents us from having Boethius theses. Dummett comments above suggested unifying both a classical negation for two kinds of truth while at the same time preserving Boethius theses, and these requirements cannot be met in the case of CN. The reader should note that Boethius theses were crucial in proving the twist representation results above (see [12]). New conditions involving the negation should be satisfied in order for the isomorphism from Theorem 4 to work properly in this case.

References

1. Cantwell, J.: The logic of conditional negation. Notre Dame Journal of Formal Logic 49(3): 245-260 (2008). https://doi.org/10.1215/00294527-2008-010
2. Cignoli, R.: The class of Kleene algebras satisfying an interpolation property and Nelson algebras. Algebra Universalis **23**, 262–292 (1986). https://doi.org/https://doi.org/10.1007/BF01230621
3. Cooper, W.S.: The Propositional Logic of Ordinary Discourse. Inquiry: An Interdisciplinary Journal of Philosophy **11**(1-4), 295–320 (1968). https://doi.org/10.1080/00201746808601531
4. Égré, P., Rossi, L., Sprenger, J.: De Finettian logics of indicative conditionals part I: trivalent semantics and validity. Journal of Philosophical Logic **50**(2), 187–213 (Apr 2021). https://doi.org/10.1007/s10992-020-09549-6
5. Egré, P., Rossi, L., Sprenger, J.: De Finettian logics of indicative conditionals part ii: proof theory and algebraic semantics. Journal of Philosophical Logic **50**(2), 215–247 (Apr 2021). https://doi.org/10.1007/s10992-020-09572-7
6. Farrell, R.J.: Material implication, confirmation, and counterfactuals. Notre Dame Journal of Formal Logic 20 (2), pp. 383-394 (1979)
7. Fazio, D., Odintsov, S.P.: An algebraic investigation of the Connexive Logic C. Studia Logica (Jun 2023). https://doi.org/10.1007/s11225-023-10057-2
8. de Finetti, B.: La logique de la probabilité. Actes du Congrés International de Philosophie Scientifique (1936)
9. Greati, V., Marcelino, S., Rivieccio, U.: Axiomatizing the logic of ordinary discourse. Proceeding of IPMU24 (to appear) (May 5 2024)
10. Greati, V., Marcelino, S., Muñoz Pérez, M., Rivieccio, U.: Analytic calculi for logics of indicative conditionals. In: Pozzato, G. L., Uustalu, T. (eds) Automated Reasoning with Analytic Tableaux and Related Methods. TABLEAUX 2025. Lecture Notes in Computer Science(), vol 15980. Springer, Cham. (2026)
11. Jung, A., Rivieccio, U.: Priestley duality for bilattices. Studia Logica 100 (1-2):223-252 (2012)
12. Muñoz Pérez, M.: Four-valued logics of indicative conditionals. Preprint. Available at https://arxiv.org/abs/2510.21882 (2025)
13. Muñoz Pérez, M., Rivieccio, U.: Indicative conditionals: Some algebraic considerations. In Kozen, D. & de Queiroz, R. (eds) Logic, Language, Information, and Computation, WoLLIC 2025, Lecture Notes in Computer Science, vol 15942, Springer (2025)
14. Rivieccio, U.: Implicative twist-structures. Algebra Univers. 71, 155–186 (2014)
15. Rivieccio, U.: Representation of De Morgan and (Semi-)Kleene lattices. Soft Computing, 24, pp. 8685–8716 (2020). https://doi.org/https://doi.org/10.1007/s00500-020-04885-w
16. Rivieccio, U.: The algebra of ordinary discourse. On the semantics of Cooper's logic. Archive for Mathematical Logic (2025). https://doi.org/10.1007/s00153-024-00961-2

Dynamic Fuzzy Language for Labeled Fuzzy Reactive Graphs

Suene C. Duarte[1][0009−0003−2418−8079], Manuel A. Martins[2], Daniel Figueiredo[2], and Regivan H. Santiago[3][0000−0002−4991−9603]

[1] Universidade Federal Rural do Semi-Árido, Brasil
[2] CIDMA – Universidade de Aveiro, Portugal
[3] Universidade Federal do Rio Grande do Norte, Brasil
suenecampos@ufersa.edu.br
regivan@dimap.ufrn.br
martins@ua.pt
daniel.figueiredo@ua.pt

Abstract. A Labeled Fuzzy Reactive Graph (LFRG) extends fuzzy graphs with labels – representing actions between states – and higher-order edges, that are responsible for updating the fuzzy values after each state transition. These structures present advantages such as being well-suited for modeling dynamic systems with path-dependencies, as well as offering a compact representation for several models that would otherwise be infinite. Introducing a dynamic fuzzy logic language for LFRGs enables formal verification of their properties, supporting applications in complex and adaptive domains.

Keywords: Labeled Fuzzy Reactive Graph · Dynamic Fuzzy Logic Language · Fuzzy Graphs.

1. Introduction

The notion of Reversal Fuzzy Reactive Graphs (RFRGs) was introduced by Campos et al. [7] and describes Fuzzy Reactive Graphs [17] enriched with high order-arrows that activate or deactivate arrows in this structure. Such graphs allow us to incorporate reactivity into the model by running modifications to the set of active edges when arrows are traversed. This capability is especially relevant in multi-agent scenarios, where the actions of one agent can influence or alter the behavior of others. Such interactions are common in environments with limited resources or involving some form of competition among agents. Although similar structures have been explored in previous studies, as first introduced in [10], they were applied within a context without fuzziness. Some authors such as Areces [1], van Benthem [4] and Marcelino & Gabbay [12] provide foundational contributions to works focused on graph-like structures whose set of edges can be changed.

In 2024, Campos et.al [6] expand the range of application for RFRG by introducing Labeled Fuzzy Reactive Graphs (LFRG) which are graphs enriched with labels on edges and high order-arrows, that activate or deactivate arrows in

this structure. In all these types of graphs, reactivity is worked between states through the action of aggregation functions. Also in [6], the operations based on aggregations of union, intersection and threshold of LFRGs and an application were presented. This paper complements [6] presenting a formal dynamic fuzzy language to verify properties of such structures.

The paper is organized as follows: Section 2 presents some basic concepts of LFRGs. Section 3 presents a formal dynamic fuzzy language to LFRGs and, finally, Section 4 provides some conclusions, final remarks and future work.

2. Preliminaries

We presume that the reader is already familiar with the fundamentals of fuzzy set theory and aggregation functions. We emphasize, however, that LFRG are based on fuzzy graphs as described in [15], where nodes do not have fuzzy values. Also, we assume that the membership function is valued in the complete lattice $[0,1] \times \{\text{ON}, \text{OFF}\}$ using the product order where $\text{OFF} \leq \text{ON}$. The membership function will be treated as equivalent to the corresponding fuzzy set.

Note 1. Since T-norms and T-conorms are associative and commutative, they can be uniquely extended to any arity. Let $J = (j_i)_{i \in I}$ be a family elements in $[0,1]$. For a T-norm T, the extension of T applied to J is denoted by $\underset{i \in J}{T} j_i$. The same notation is applied to any T-conorm S.

The extensions of T and S are such that they become the identity function case I is a singleton, and $\underset{i \in J}{T} j_i = 1$, $\underset{i \in J}{S} j_i = 0$ case $I = \{\}$.

We briefly recall the basic notions about Labeled Fuzzy Reactive Graphs, as introduced in [6]. In the paper, the authors consider that, under certain conditions, more than one edge can be crossed at the same time. In what follows, we present this structure and describe how the model updates occur.

Definition 1 (Labeled Fuzzy Reactive Graph [6]). *Let $\mathcal{L}$ be a set of **labels** and A a set of **ternary aggregations**. A **Labeled Fuzzy Reactive Graph** (LFRG) is an structure $M = \langle W, S^{\mathcal{L}}, \mu, Ag_M \rangle$ s.t.*

- *W is a non-empty set of states/nodes;*
- *$S^{\mathcal{L}} = \bigcup_{i \geq 0} S_i^{\mathcal{L}}$ is a set of generalized and labeled edges where:*
  - *$S_0^{\mathcal{L}} \subseteq W \times \mathcal{L} \times W$; and $S_n^{\mathcal{L}} \subseteq S_0^{\mathcal{L}} \times S_{n-1}^{\mathcal{L}} \times \{\circ, \bullet\}$;*
- *$\mu : S^{\mathcal{L}} \to [0,1] \times \{\text{ON}, \text{OFF}\}$ is a fuzzy membership function;*
- *$Ag_M : S_{\to}^{\mathcal{L}} \longrightarrow A$, where*

  $$S_{\to}^{\mathcal{L}} := \left\{ a_i^0 \in S_0^{\mathcal{L}};\ [\![a_i^0, b, \sigma]\!] \in S^{\mathcal{L}} \text{ for some } b \in S^{\mathcal{L}} \text{ and some } \sigma \in \{\circ, \bullet\} \right\}$$

  is the set of source arrows.

For any set $S \subseteq S^{\mathcal{L}}$, $S^* = \{s \in S : \mu_2(s) = \text{ON}\}$ *contains its active arrows.*

To prevent ambiguity, we impose that $(a, b, \circ) \notin S_n^{\mathcal{L}}$ *or* $(a, b, \bullet) \notin S_n^{\mathcal{L}}$, *for* $n \geq 1$.

Arrows that have $\bullet$ in the third component are called connecting arrows and that have $\circ$ in the third component are called disconnecting arrows. Graphically, we represent active arrows with a solid line, while inactive arrows are drawn as dashed. The process of connecting (resp. disconnecting) arrows modifies the state of the target arrow into active (resp. inactive), and these actions are indicated with a black (resp. white) arrowhead.

For LFRG, the case in which a connecting and disconnecting arrow, with the same source arrow, acting over the same target arrow is allowed only if even one being ON and the other OFF and they have different labels.

Example 1. In Fig.2, we get $\mathcal{L} = \{\alpha, \beta, \gamma, \theta\}$, $S_0^{\mathcal{L}} = \{[x, \alpha, y], [y\beta, z], [x, \gamma, z], [x, \theta, w]\}$ and $S_1^{\mathcal{L}} = \{[\![[y\beta, z], [x, \gamma, z], \circ]\!], [\![[x, \gamma, z], [x, \theta, w], \bullet]\!]\}$. The arrow $[\![[y\beta, z], [x, \gamma, z], \circ]\!]$ is a disconnecting arrow and $[\![[x, \gamma, z], [x, \theta, w], \bullet]\!]$ is a connecting arrow. The Table 1 shows the values of the arrows applied to the μ function.

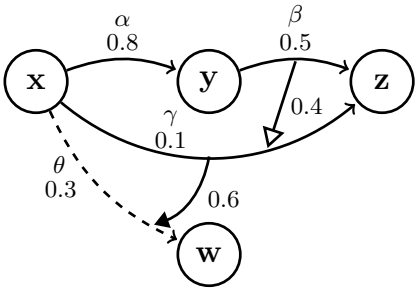


Fig. 1: Example of a LFRG

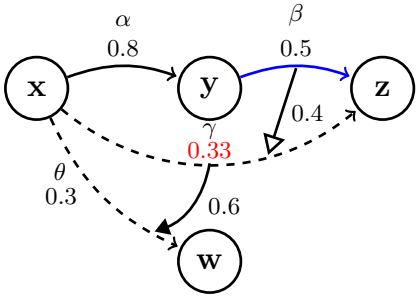


Fig. 2: LFRG after cross the arrow $[y, \beta, z]$

Arrow	μ
$[x, \alpha, y]$	(0.8, ON)
$[y\beta, z]$	(0.5, ON)
$[x, \gamma, z]$	(0.1, ON)
$[x, \theta, w]$	(0.3, OFF)
$[\![[y\beta, z], [x, \gamma, z], \circ]\!]$	(0.4, ON)
$[\![[x, \gamma, z], [x, \theta, w], \bullet]\!]$	(0.6, ON)

Table 1: Image of the fuzzy μ function

In order to avoid inconsistency, [6] establishes that a set of high-order arrows pointing to an arrow b , if crossed simultaneously, will only exert modifications on b if all are connection arrows or, all are disconnection arrows.

Fig.2 shows the LFRG of Fig.1(b) after crossing $[y, \beta, z]$ and interpreting $Ag_M([y, \beta, z])$ as the arithmetic mean. For other examples, considering even the case of several arrows being crossed simultaneously, see [6].

3. A Dynamic Fuzzy Language for LFRG

In order to present a logical language to verify properties of a system modeled by a LFRG, we propose a dynamic logic, which takes into account the ones proposed in [3] and [17], to verify the properties of Fuzzy Reactive Graphs. Thus, we provide a formal language which combines both Propositional Dynamic Logic (PDL) and fuzzy semantics. PDL is one of the most prominent applied modal logics and enables applications in different areas [11, 13, 18]. In this paper, we consider LFRG with a finite number of states and labels.

Given the fact that we can cross more than one (parallel) edge on an RFSG, we consider a dynamic logic with a concurrent composition (or intersection) operator, which is based on [3]. However, we note that if we only allow a single edge to be crossed at each time, we do not need it, and we can thus consider the fragment of this logic without the concurrent composition operator.

The language of a propositional dynamic fuzzy logic for LFRG consider a set *AProp* of atomic propositions, the set $\mathcal{L}$ of labels and the following symbols:

- logical symbols: $\top, \bot, \neg, \vee, \wedge, \Rightarrow, \Leftrightarrow$;
- label symbols: ; (composition), + (nondeterministic choice) and $\wedge$ (concurrent composition or intersection) and ? (test);
- parentesis: (,).

We use the symbol $\wedge$ for logical conjunction, and concurrent composition of labels. However, its meaning should be clear according the context in which they are used.

Definition 2 (Syntax). *Consider the set AProp of all propositions and the set $\mathcal{L}$ of all labels. The set Form of formulas and the set $\mathbb{L}$ of words are defined as the smallest sets such that:*

- $AProp \subseteq Form$ *and* $\mathcal{L} \subseteq \mathbb{L}$;
- *If* $\varphi, \psi \in Form$, *then* $\bot$, $\top$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi \Rightarrow \psi$, $\varphi \Leftrightarrow \psi$, $\neg\varphi \in Form$;
- *If* $a_1, ..., a_n \in \mathcal{L}$ *with* $n > 1$, *then* $\wedge(a_1, ..., a_n) \in \mathbb{L}$
- *If* $\alpha, \beta \in \mathbb{L}$, *then* $\alpha;\beta$, $\alpha + \beta \in \mathbb{L}$
- *If* $\varphi \in Form$ *and* $\alpha \in \mathbb{L}$, *then* $[\alpha]\varphi$, $\langle\alpha\rangle\varphi \in Form$;
- *If* $\varphi \in Form$, *then* $\varphi? \in \mathbb{L}$.

Definition 3. *A **Labeled Fuzzy Reactive Model** (LFRM) is a tuple $\mathcal{M} = \langle M, V, \mathcal{H}\rangle$ s.t. $M = \langle W, S^{\mathcal{L}}, \mu, Ag_M\rangle$, $V : W \times AProp \longrightarrow [0,1]$ is a fuzzy valuations and $\mathcal{H} = \left(H^j : [0,1]^j \to [0,1]\right)_{2\leq j\leq m}$ a family of symmetric aggregations.*

The family of aggregations $\mathcal{H}$ is only needed when more than one arrow is travessed simultaneously to aggregate their impact on target arrows (check [6]).

Definition 4 (Satisfaction). *Let $\mathcal{F} = \langle [0,1], T, S, N, I, B, 0, 1\rangle$ be a fuzzy semantic and $\mathcal{M} = \langle M, \mathcal{H}, V\rangle$ be a LFRM. We define the grade of certainty of a given formula φ to be true at state w, $[\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w}$, recursively:*

- $[\![p]\!]^{\mathcal{F}}_{\mathcal{M},w} = V(w,p)$, *for* $p \in AProp$;
- $[\![\neg\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \mathrm{N}([\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w})$;
- $[\![\varphi \wedge \psi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \mathrm{T}([\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w}, [\![\psi]\!]^{\mathcal{F}}_{\mathcal{M},w})$;
- $[\![\varphi \Rightarrow \psi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \mathrm{I}([\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w}, [\![\psi]\!]^{\mathcal{F}}_{\mathcal{M},w})$;
- $[\![\top]\!]^{\mathcal{F}}_{\mathcal{M},w} = 1$;
- $[\![\bot]\!]^{\mathcal{F}}_{\mathcal{M},w} = 0$;
- $[\![\varphi \vee \psi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \mathrm{S}([\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w}, [\![\psi]\!]^{\mathcal{F}}_{\mathcal{M},w})$;
- $[\![\varphi \Leftrightarrow \psi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \mathrm{B}([\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w}, [\![\psi]\!]^{\mathcal{F}}_{\mathcal{M},w})$

- $[\![[\beta]\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \underset{w' \,:\, (w,\beta,w')\in(S_0^{\mathcal{L}})^*}{T}\Big(\mathrm{I}\big(\mu_1(w,\beta,w'), [\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M}^{Ag_M}_{(w,\beta,w')},w'}\big)\Big)$,
  where $\mathcal{M}^{Ag_M}_{(w,\beta,w')} = \big(M^{Ag_M}_{(w,\beta,w')}, \mathcal{H}, V\big)$;
- $[\![\langle\beta\rangle\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \underset{w' \,:\, (w,\beta,w')\in(S_0^{\mathcal{L}})^*}{S}\Big(\mathrm{T}\big(\mu_1(w,\beta,w'), [\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M}^{Ag_M}_{(w,\beta,w')},w'}\big)\Big)$,
  where $\mathcal{M}^{Ag_M}_{(w,\beta,w')} = \big(M^{Ag_M}_{(w,\beta,w')}, \mathcal{H}, V\big)$;
- $[\![[\wedge(\alpha_1,...,\alpha_n)]\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \underset{\substack{w' \,:\forall\beta\in\{\alpha_1,\ldots,\alpha_n\},\\ (w,\beta,w')\in(S_0^{\mathcal{L}})^*}}{T}\Big(\mathrm{I}\big(\mu_1(w,\beta,w'), [\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M}^{Ag_M,\mathcal{H}}_{\bar{a}},w'}\big)\Big)$,
  where $\mathcal{M}^{Ag_M,\mathcal{H}}_{\bar{a}} = \big(M^{Ag_M,\mathcal{H}}_{\bar{a}}, \mathcal{H}, V\big)$ *with* $\bar{a}$ *being a sequence of the arrows contained in* $\{(w,\beta,w') \in (S_0^{\mathcal{L}})^* : \beta \in \{\alpha_1,...,\alpha_n\}$;
- $[\![\langle\wedge(\alpha_1,...,\alpha_n)\rangle\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \underset{\substack{w' \,:\forall\beta\in\{\alpha_1,\ldots,\alpha_n\},\\ (w,\beta,w')\in(S_0^{\mathcal{L}})^*}}{S}\Big(\mathrm{T}\big(\mu_1(w,\beta,w'), [\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M}^{Ag_M,\mathcal{H}}_{\bar{a}},w'}\big)\Big)$,
  where $\mathcal{M}^{Ag_M,\mathcal{H}}_{\bar{a}} = \big(M^{Ag_M,\mathcal{H}}_{\bar{a}}, \mathcal{H}, V\big)$ *with* $\bar{a}$ *being a sequence of the arrows contained in* $\{(w,\beta,w') \in (S_0^{\mathcal{L}})^* : \beta \in \{\alpha_1,...,\alpha_n\}$;

- $[\![[\alpha+\beta]\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \mathrm{T}\big([\![[\alpha]\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w}, [\![[\beta]\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w}\big)$;
- $[\![\langle\alpha+\beta\rangle\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \mathrm{S}\big([\![\langle\alpha\rangle\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w}, [\![\langle\beta\rangle\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w}\big)$;

- $[\![[\alpha;\beta]\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = [\![[\alpha]([\beta]\varphi)]\!]^{\mathcal{F}}_{\mathcal{M},w}$;
- $[\![[\psi?]\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \mathrm{I}([\![\psi]\!]^{\mathcal{F}}_{\mathcal{M},w}, [\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w})$;

- $[\![\langle\alpha;\beta\rangle\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = [\![\langle\alpha\rangle(\langle\beta\rangle\varphi)]\!]^{\mathcal{F}}_{\mathcal{M},w}$;
- $[\![\langle\psi?\rangle\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w} = \mathrm{T}([\![\psi]\!]^{\mathcal{F}}_{\mathcal{M},w}, [\![\varphi]\!]^{\mathcal{F}}_{\mathcal{M},w})$;

Example 2. Consider *AProp*={p} and let $\mathcal{M}$ be the LFRM illustrated by Fig. 2 where V defined according to the Table 2. We provide an example of how to evaluate the formula $\langle\alpha;\alpha\rangle p$ at the state x, which means that it is possible to reach a state from x where p is satisfied, with the word $\alpha;\alpha$.

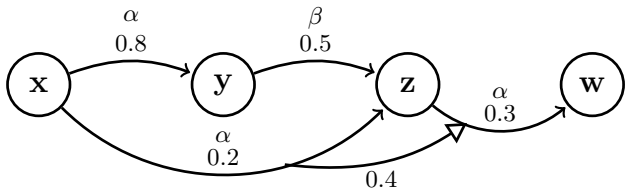


Fig. 3: Illustration of the LFRM used in Example 2.

	x	y	z	w
p	0.2	0	0.4	0.1

Table 2: Truth values of propositions on each state.

For the following evaluation, we consider Ag_M be the arithmetic mean for all zero-order arrows, the Godel Semantic $\mathcal{F}_G$ and note that we do not need to define $\mathcal{H}$, as there are no parallel edges between states.

$[\![\langle\alpha;\alpha\rangle p]\!]^{\mathcal{F}}_{\mathcal{M},x}$
$= [\![\langle\alpha\rangle(\langle\alpha\rangle p)]\!]^{\mathcal{F}}_{\mathcal{M},x}$

$$= \mathrm{S_G}\Big(\mathrm{T_G}\big(0.8, [\![\langle\alpha\rangle p]\!]^{\mathcal{F}}_{\mathcal{M}^{Ag_M}_{(x,\alpha,y)},y}\big),\ \mathrm{T_G}\big(0.2, [\![\langle\alpha\rangle p]\!]^{\mathcal{F}}_{\mathcal{M}^{Ag_M}_{(x,\alpha,z)},z}\big)\Big)$$

(In this next step, we are evaluating the formula not only in different states but also in different models)

$$= \mathrm{S_G}\Big(\mathrm{T_G}\big(0.8, [\![\langle\alpha\rangle p]\!]^{\mathcal{F}}_{\mathcal{M},y}\big),\ \mathrm{T_G}\big(0.2, [\![\langle\alpha\rangle p]\!]^{\mathcal{F}}_{\mathcal{M}^{Ag_M}_{(x,\alpha,z)},z}\big)\Big)$$
$$= \mathrm{S_G}\Big(\mathrm{T_G}\big(0.8, 0\big)\ ,\ \mathrm{T_G}\big(0.2, 0\big)\Big)$$
$$= \ \mathrm{S_G}(0, 0)$$
$$= 0$$

In this example, $[\![\langle\alpha\rangle p]\!]^{\mathcal{F}}_{\mathcal{M},y}$ is evaluated as 0 due to what is established for t-conorm is Note 1. The same reasoning is used to $[\![\langle\alpha\rangle p]\!]^{\mathcal{F}}_{\mathcal{M}^{Ag_M}_{(x,\alpha,z)},z}$, taking into account that $\mu \neq \mu^{Ag_M}_{(x,\alpha,z)}$. The arrow $[z, \alpha, w]$ is such that $\mu^{Ag_M}_{(x,\alpha,z)}([z, \alpha, w] = \textsc{off}$.

Moreover, note that if we had replace the word within the modality by $\alpha; (p?+\beta)$, we would need to evaluate the same proposition p at the same state z, but in different models (one for each choice in the nondeterministic operator $+$). This is one example where using a word-parameterized accessibility relation $R_{\alpha;(p?+\beta)} : W \times W \to [0, 1]$ would not be feasible. Indeed, if instead of p we had a more complex formula (such as $[\alpha]\bot$), this formula would be satisfied at z in one of the models but not in the other.

4. Conclusion and future work

Labeled Fuzzy Reactive Graphs (LFRG) are structures designed to model reactive systems which provide the activation and deactivation of resources with actions assigned to zero-order edges. This paper proposes a fuzzy propositional dynamic language for LFRG with concurrent composition (intersection) to permit one to check properties of systems modeled by them. The paper also provides some examples.

The proposed logic aims to generalize both the classical PDL and the one proposed in [17] for Fuzzy Reactive Graphs, as these can be understood as a particular case of LFRG. However, the PDL closure (also known as iteration) operator, commonly represented by *, was not included in the syntax. As noted in [17], the fuzzy reconfigurable structures can be used to finitely represent infinite structures. As the use of the closure operator could require the consideration of all possible reconfigurations of a model, this would lead to a possibly infinite process.

Finally, we would like to apply this model to real-life cases with multi-agents and resource consumption scenarios. To ease the application of these tools, we would like to develop a computational implementation of an automatic checker.

Acknowledgments. This work is supported by FCT – Fundação para a Ciência e a Tecnologia through projects UIDB/04106/2025 at CIDMA and by National and European Funds through SACCCT- IC&DT - Sistema de Apoio à Criação de Conhecimento Científico e Tecnológico, as part of COMPETE2030, within the project BANKSY with reference number 15253.

References

1. C. Areces, R. Fervari and G. Hoffmann, *Relation-changing modal operators*. Logic Journal of the IGPL 23(4), 601-627, 2015. https://doi.org/10.1093/jigpal/jzv020.
2. M. Baczyński and B. Jayaram, *An Introduction to Fuzzy Implications*, Studies in Fuzziness and Soft Computing 231, 1-35, 2008. https://doi.org/10.1007/978-3-540-69082-5_1
3. P. Balbiani and D. Vakarelov, *Iteration-free PDL with Intersection: a Complete Axiomatization*, Fundamenta Informatica 45(3), 173-194, 2001. https://doi.org/10.3233/FUN-2001-45302
4. J. van Benthem, *An Essay on Sabotage and Obstruction*, Lecture Notes in Computer Science 2605, 268-276, Springer, 2005. https://doi.org/10.1007/978-3-540-32254-216.
5. C. Callejas, J. Marcos and B. Bedregal, *Actions of automorphisms on some classes of fuzzy bi-implications*, Mathware & Soft Computing magazine 177, 140-146, 2012.
6. S. Campos, D. Figueiredo, M. A. Martins, and R. Santiago, *Labeled fuzzy reactive graphs*, Fuzzy Sets and Systems 510, 109320, 2025. https://doi.org/10.1016/j.fss.2025.109320.
7. S. Campos, R. Santiago, M.A. Martins and D. Figueiredo, *Introduction to reversal fuzzy switch graph*, Science of Computer Programming 216, 102776, 2022. https://doi.org/10.1016/j.scico.2022.102776
8. S. Campos, R. Santiago, M. A. Martins and D. Figueiredo, *Introduction to reversal fuzzy switch graph*, Science of Computer Programming 216, 102776, 2022. https://doi.org/10.1016/j.scico.2022.102776.
9. A. Cruz, B. Bedregal and R. Santiago, *On the characterizations of fuzzy implications satisfying* $i(x, i(y, z)) = i(i(x, y), i(x, z))$, International Journal of Approximate Reasoning 93, 261-276,2018. https://doi.org/10.1016/j.ijar.2017.11.004
10. D. Figueiredo, M. A. Martins and L. S. Barbosa, *A Note on Reactive Transitions and Reo Connectors*, It's All About Coordination, Lecture Notes in Computer Science 10865, 57-67, 2018. https://doi.org/10.1007/978-3-319-90089-64
11. M. Fisher and R. Ladner, *Propositional dynamic logic of regular programs*, Journal of Computer and System Sciences 18(2), 194-211, 1979. https://doi.org/10.1016/0022-0000(79)90046-1
12. D. M. Gabbay and S. Marcelino, *Global view on reactivity: switch graphs and their logics*, Annals of Mathematics and Artificial Intelligence 66, 131–162, Springer, 2012. https://doi.org/10.1007/s10472-012-9316-8.
13. D. Harel, *Dynamic logic*, Handbook of Philosophical Logic, Synthese Library 165, 497–604, 1984. https://doi.org/10.1007/978-94-009-6259-0_10
14. E. Klement, R. Mesiar and E. Pap, Triangular Norms, Springer, 2013. https://doi.org/10.1007/978-94-015-9540-7
15. K. H. Lee, *First course on fuzzy theory and applications*. Advances in Intelligent and Soft Computing 27, Springer, 2005. https://doi.org/10.1007/3-540-32366-X.
16. A. Madeira, R. Neves, and M. A: Martins, *An exercise on the generation of many-valued dynamic logics*, Journal of Logical and Algebraic Methods in Programming 85(5), 1011-1037, 2016. https://doi.org/10.1016/j.jlamp.2016.03.004

17. R. Santiago, M. A. Martins and D. Figueiredo, *Introducing fuzzy reactive graphs: a simple application on biology*, Soft Computing 25, 6759–6774, 2021. https://doi.org/10.1007/s00500-020-05353-1.
18. K. Segerberg, *A completeness theorem in the modal logic of programs*, A completeness theorem in the modal logic of programs 9, 31–36, 1982. http://eudml.org/doc/209235