EPTCS 390

Proceedings of the

Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification

Udine, Italy, 18-20th September 2023

Edited by: Antonis Achilleos and Dario Della Monica

Published: 30th September 2023
DOI: 10.4204/EPTCS.390
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Preface

This volume contains the proceedings of GandALF 2023, the Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification. The symposium was held in Udine, Italy, on September 18-20, 2023.

The GandALF symposium was established by a group of Italian computer scientists to provide an opportunity for researchers interested in logic for computer science, automata theory, game theory, to gather and discuss the application of formal methods to the specification, design, and verification of complex systems. Previous editions of GandALF were held in Madrid, Spain (2022); Padova, Italy (2021); Brussels, Belgium (2020); Bordeaux, France (2019); Saarbrücken, Germany (2018); Rome, Italy (2017); Catania, Italy (2016); Genoa, Italy (2015); Verona, Italy (2014); Borca di Cadore, Italy (2013); Napoli, Italy (2012); and Minori, Italy (2011 and 2010). It is a forum where people from different areas, and possibly with different backgrounds, can fruitfully interact, with a truly international spirit, as witnessed by the composition of the program and steering committees and by the country distribution of the submitted papers.

The program committee selected 15 papers (out of 26 submissions) for presentation at the symposium. Each paper was reviewed by at least three referees, and the selection was based on originality, quality, and relevance to the topics of the call for papers. The scientific program included presentations on automata and automata learning, logics for computer science and verification, computational and descriptive complexity theory, formal methods and specification languages, concurrency and process semantics, games, strategic reasoning, and synthesis. The program included four invited talks, given by Laure Daviaud (University of East Anglia, UK), Juha Kontinen (University of Helsinki, Finland), Sophie Pinchinat (IRISA/University of Rennes, France), and Alexander Rabinovich (Tel Aviv University, Israel). We are deeply grateful to them for contributing to this year edition of GandALF.

We would like to thank the authors who submitted papers for consideration, the speakers, the program committee members and the additional reviewers for their excellent work. We also thank EPTCS and arXiv for hosting the proceedings; in particular, we thank Rob van Glabbeek for the precise and prompt technical support with issues related with the proceeding publication procedure.

Finally we would like to thank the local organisers: Andrea Brunello, Renato Acampora, Luca Geatti, Nicola Gigante, Mattia Guiotto, Gabriele Puppis, and Nicola Saccomanno for making sure the event ran smoothly.

Antonis Achilleos and Dario Della Monica

Program Chairs

- Dario Della Monica, University of Udine (Italy)
- Antonis Achilleos, Reykjavik University (Iceland)

Program Committee

- Parosh Aziz Abdulla, Uppsala University (Sweden)

Steering Committee

External Reviewers

Complexity Aspects of Logics in Team Semantics

Juha Kontinen (University of Helsinki, Finland)

Team Semantics is the mathematical basis of modern logics for reasoning about dependence, independence, and imperfect information. During the past decade research on team semantics has flourished with interesting connections to fields such as database theory, statistics, formal linguistics, hyperproperties and causality, just to mention a few examples. I will give a short introduction to first-order team semantics and review the expressivity and complexity of the most prominent logics of the area. I will also discuss probabilistic variants of these logics and their connections to the existential theory of the reals.

Weighted Automata at the Border of Decidability

Laure Daviaud (University of East Anglia, UK)

In this talk I will review some results about weighted automata: what is decidable (or not) when it comes to describing their behaviour and what are the mathematical tools that are used to prove such results. In particular, I will describe Simon's factorisation theorem, how it plays a major role in several of these results and how it could be used more widely.

The Church Synthesis Problem Over Continuous Time

Alexander Rabinovich and Daniel Fattal (Tel-Aviv University, Israel)

Church's Problem asks for the construction of a procedure which, given a logical specification $\varphi(I,O)$ between input ω-strings I and output ω-strings O, determines whether there exists an operator F that implements the specification in the sense that $\varphi(I,F(I))$ holds for all inputs I. Büchi and Landweber gave a procedure to solve Church's problem for MSO specifications and operators computable by finite-state automata. We investigate a generalization of the Church synthesis problem to the continuous time of the non-negative reals. It turns out that in the continuous time there are phenomena which are very different from the canonical discrete time domain of the natural numbers.

Strategic Reasoning under Imperfect Information – The Case of Synchronous Recall

Sophie Pinchinat (IRISA/University of Rennes, France)

We first briefly survey various formal settings for Strategic Reasoning under Imperfect Information in multi-player games. From the point of view of automated verification and/or synthesis, we motivate the assumption of Synchronous Recall, that enforces particular properties on players' observation capabilities.

Second, with the focus on aforementioned Synchronous Recall assumption, we consider arenas that arise from Dynamic Epistemic Logic, a modal logic dedicated to specifying information changes in a multi-player game. Such arenas are described in a symbolic manner, with an initial epistemic state and rules for player moves to attain new epistemic state, thus reflecting the information change that takes place.

We then explain how such symbolic arenas denote a possibly infinite first-order structure - reminiscent of the trees considered for interpreting Epistemic Temporal Logic. Second, we exploit this angle of view to go through the current state-of-the-art for reasoning (including planning) about these structures, and analyse the frontier of this overall setting regarding Strategic Reasoning.

A Uniform One-Dimensional Fragment with Alternation of Quantifiers

Emanuel Kieroński
Institute of Computer Science
University of Wrocław, Poland
emanuel.kieronski@cs.uni.wroc.pl

The uniform one-dimensional fragment of first-order logic was introduced a few years ago as a generalization of the two-variable fragment of first-order logic to contexts involving relations of arity greater than two. Quantifiers in this logic are used in blocks, each block consisting only of existential quantifiers or only of universal quantifiers. In this paper we consider the possibility of mixing quantifiers in blocks. We identify a non-trivial variation of the logic with mixed blocks of quantifiers which retains some good properties of the two-variable fragment and of the uniform one-dimensional fragment: it has the finite (exponential) model property and hence decidable, NExpTime-complete satisfiability problem.

1 Introduction

In this paper we are going to push forward the research on the uniform one-dimensional fragment of first-order logic. To set up the scene and locate our results in a broader context let us first recall some facts about the two-variable fragment, FO^2. FO^2, obtained just by restricting first-order logic so that its formulas may use only variables x and y, is one of the most important decidable fragments of first-order logic identified so far. The decidability of its satisfiability problem was shown by Scott [34] in the case without equality, and by Mortimer [26] in the case with equality. In [26] it is proved that the logic has the finite model property, that is, its every satisfiable formula has a finite model. Later, Grädel, Kolaitis and Vardi [11] strengthened that result, by showing that every satisfiable formula has a model of size bounded exponentially in its length. This exponential model property led to the NExpTime upper bound on the complexity of FO^2 satisfiability. The matching lower bound follows from the earlier work by Lewis [24].

An important motivation for studying FO^2 is the fact that it embeds, via the so-called standard translation, basic modal logic and many standard description logics. Thus FO^2 constitutes an elegant first-order framework for those formalisms. However, its simplicity and naturalness make it also an attractive logic in itself, inheriting potential applications in knowledge representation, artificial intelligence, or verification of hardware and software from modal and description logics. Plenty of results on FO^2, its extensions and variations have been obtained in the last few decades, e.g., decidability was shown for FO^2 with counting quantifiers [12, 27, 28], one or two equivalence relations [19, 18], counting quantifiers and equivalence relation [29], betweenness relations [22], its complexity was established on words and trees, in various scenarios including the presence of data or counting [5, 4, 8, 9, 3], to mention just a few of them.

However, further applications, e.g., in database theory are limited by the fact that FO^2 and its extensions mentioned above can speak non-trivially only about relations of arity at most two. This is in contrast to some other decidable fragments studied because of their potential applications in computer science, like the guarded fragment, GF [1], the unary negation fragment, UNFO [7], the guarded negation fragment, GNFO [2], or the fluted fragment, FF [32, 31].

A. Achilleos and D. Della Monica (Eds.): Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification (GandALF 2023).
EPTCS 390, 2023, pp. 1–15, doi:10.4204/EPTCS.390.1

A natural question is whether there is an elegant decidable formalism which retains full expressivity of FO^2, but additionally, allows one to speak non-trivially about relations of arity bigger than two. In the recent literature we can find a few such formalisms.

An interesting idea is for example to combine FO^2 with GF. The idea can be traced back already in Kazakov's PhD thesis [15], was present in the work by Bourish, Morak and Pieris [6], and found a more systematic treatment in the paper by Rudolph and Šimkus [33], who formally introduced the triguarded fragment, TGF. TGF is obtained from GF by allowing quantification for subformulas with at most two free variables to be unguarded. What we get this way is a logic in which one can speak freely about pairs of elements, and in a local, guarded way about tuples of bigger arity. TGF turns out to be undecidable with equality, but becomes decidable when equality is forbidden. The satisfiability problem is then 2-ExpTime- or 2-NExpTime-complete, depending on whether constants are allowed in signatures [33]; the finite model property is retained [21]. A variation of the idea above is the one-dimensional triguarded fragment [20], still containing FO^2, which becomes decidable even in the presence of equality.

FO^2 (or, actually, even its extension with counting quantifiers, C^2) was also combined with FF by Pratt-Hartmann [30]. This logic was shown decidable but the complexity of its satisfiability problem is non-elementary, as already FF alone has non-elementary complexity [31].

Finally, probably the most canonical extension of FO^2 to contexts with relations of arity bigger than two, capturing the spirit of FO^2 more closely than the logics discussed above, is the *uniform one-dimensional fragment*, UF$_1$, proposed by Hella and Kuusisto [13]. In this fragment quantifiers are used in blocks and a single block is built out only of existential or only of universal quantifiers and leaves at most one variable free; a fragment meeting this condition is called *one-dimensional*. Imposing one-dimensionality alone is not sufficient for ensuring the decidability of the satisfiability problem and thus another restriction, *uniformity*, is applied which, roughly speaking, allows boolean combinations of atoms only if the atoms use precisely the same set of variables or use just one variable. In effect, just as FO^2 contains modal logic (or even Boolean modal logic), UF$_1$ contains *polyadic* modal logic (even with negations of the accessibility relations) (cf. [23]). In [13] it is shown that UF$_1$ without equality is decidable and has the finite model property. In [16] this result is improved by showing that the decidability is retained even if free use of equalities is allowed (by *free use* of equalities we mean that they need not obey the uniformity restriction) and that the logic has exponential model property and NExpTime-complete satisfiability problem, exactly as FO^2.

A question arises whether the requirement that the blocks of quantifiers from the definition of UF$_1$ must consist of quantifiers of the same type (all universal or all existential) is necessary for decidability, that is what happens if we allow one to mix quantifiers as, e.g., in the formula $\forall x \exists y \forall z R(x,y,z,t)$. Let us denote the extension of UF$_1$ allowing to alternate quantifiers in blocks AUF$_1$. The motivations behind studying AUF$_1$ are multifarious. UF$_1$ lies very close to the borderlines between the decidable and undecidable, so, firstly and most importantly, analysing its expressive extensions may enhance our understanding of these borderlines which may be also useful in different scenarios. Secondly, the logics UF$_1$ and AUF$_1$ can be useful themselves, offering extensions of modal and description logics to contexts with relations of arity greater than two, such as databases, orthogonal to other proposals. Thirdly, though it is of course a matter of taste, we believe that AUF$_1$ is just quite an elegant formalism, which can be justified by a relative simplicity of its definition and a nice game-theoretic characterization of its expressivity—natural Ehrenfeucht-style games for UF$_1$ were introduced in [16]; shifting to AUF$_1$ would probably allow for an even nicer game characterizations (though this topic is not formally studied in this paper).

The first step to understand AUF$_1$ was done in the companion paper [10], where we show the decidability and the finite model property of the three variable restriction of this logic, AUF$_1^3$; in that paper

AUF_1^3 is then made a basis for obtaining a rich decidable subclass of the three-variable fragment, FO^3.

Turning now to our current contribution, we first remark that in this paper we still do not answer the question whether the whole AUF_1 has decidable satisfiability. We however make another step towards understanding AUF_1 by identifying its fragment, AUF_1^-, which contains full FO^2 without equality, allows for mixed blocks of quantifiers of unbounded length, has NExpTime-complete satisfiability problem, and has the exponential model property. Additionally, we observe that if we allow for a free use of equality in AUF_1^- then we lose the finite model property.

The main restriction of AUF_1^-, compared to full AUF_1, is that it admits only blocks of quantifiers that are purely universal or end with the existential quantifier. Additionally, mostly for the clarity of presentation, we will define AUF_1^- not as an extension of the version of UF_1 originally defined in [13], but rather as an extension of the *strongly* uniform one-dimensional fragment, sUF_1, introduced in [17]. The definition of AUF_1^- is inspired by the definition of the Maslov class $\overline{\mathrm{K}}$ [25] and, as we will see in a moment, the decidability of AUF_1^- can be shown by a reduction to conjunctions of sentences in $\overline{\mathrm{K}}$, whose decidability was shown by the resolution method by Hustadt and Schmidt [14]. However, this reduction does not allow us to establish the precise compleixty of AUF_1^-, since, to the best of our knowledge, the precise complexity of the Maslov class has not been established. It is also not known whether $\overline{\mathrm{K}}$ has the finite model property.

2 Preliminaries

2.1 Notation and terminology

We assume that the reader is familiar with first-order logic. We work with purely relational signatures with no constants nor function symbols. We refer to structures using Fraktur capital letters, and to their domains using the corresponding Roman capitals. Given a structure $\mathfrak{A}$ and some $B \subseteq A$ we denote by $\mathfrak{A}\restriction B$ the restriction of $\mathfrak{A}$ to its subdomain B.

We usually use $a, b, \ldots$ to denote elements of structures, and $x, y, \ldots$ for variables; all of these possibly with some decorations. For a tuple of variables $\bar{x}$ we use $\psi(\bar{x})$ to denote that the free variables of ψ are in $\bar{x}$.

In the context of uniform logics it is convenient to speak about some partially defined (sub)structures which we will call *pre-(sub)structures*. A pre-structure over a signature σ consists of its domain A and a function specifying the truth-value of every fact $P(\bar{a})$, for $P \in \sigma$ and a tuple $\bar{a}$ of elements of A of length equal to the arity of P, such that $\bar{a}$ contains all elements of A or just one of them. The truth values of all the other facts remain unspecified. We will use Fraktur letters decorated with $*$ to denote pre-structures: a pre-structure with domain A will be denoted by $\mathfrak{A}^*$. If a structure $\mathfrak{A}$ is fully defined, $\mathfrak{A}^*$ denotes its induced pre-structure. Similarly, if $B \subseteq A$ is a subdomain of some structure $\mathfrak{A}$ we donote by $\mathfrak{B}^*$ the pre-structure $(\mathfrak{A}\restriction B)^*$ and call it a pre-substructure of $\mathfrak{A}$.

An (atomic) 1-*type* over a signature σ is a maximal consistent set of atomic or negated atomic formulas over σ using at most one variable x. We often identify a 1-type with the conjunction of its elements. We will usually be interested in 1-types over signatures σ consisting of the relation symbols used in some given formula. Observe that the number of 1-types is bounded by a function which is exponential in $|\sigma|$, and hence also in the length of the formula. This is because a 1-type just corresponds to a subset of σ.

Let $\mathfrak{A}$ be a structure, and let $a \in A$. We denote by $\mathrm{tp}^{\mathfrak{A}}(a)$ the unique atomic 1-type *realized* in $\mathfrak{A}$ by the element a, *i.e.*, the 1-type $\alpha(x)$ such that $\mathfrak{A} \models \alpha(a)$.

2.2 Satisfiability and finite model property

Let $\mathscr{L}$ be a class of first-order formulas (a logic). The *(finite) satisfiability problem* for $\mathscr{L}$ takes as its input a sentence from $\mathscr{L}$ and verifies if it has a (finite) model. $\mathscr{L}$ has the *finite model property* if every satisfiable sentence in $\mathscr{L}$ has a finite model; $\mathscr{L}$ has the *exponential model property* if there is a fixed exponential function f such that every satisfiable sentence φ has a finite model over a domain whose size if bounded by $f(|\varphi|)$ (where the length of φ, $|\varphi|$, is measured in any reasonable fashion).

2.3 Logics

As the starting point we define the logic sUF_1 (without equality), called in [17] the *strongly restricted uniform one-dimensional fragment*. Formally, for a relational signature σ, the set of σ-formulas of sUF_1 is the smallest set $\mathscr{F}$ such that:

- every σ-atom using at most one variable is in $\mathscr{F}$
- $\mathscr{F}$ is closed under Boolean connectives
- if $\varphi(x_0,\dots,x_k)$ is a Boolean combination of formulas in $\mathscr{F}$ with free variables in $\{x_0,\dots,x_k\}$ and atoms[1] built out of precisely all of the variables $x_0,\dots,x_k$ (in an arbitrary order, possibly with repetitions) then $\exists x_0,\dots,x_k\varphi$, $\exists x_1,\dots,x_k\varphi$, $\forall x_0,\dots,x_k\varphi$ and $\forall x_1,\dots,x_k\varphi$ are in $\mathscr{F}$.

Example formulas in sUF_1 are:

$$\forall xyz(P(x) \wedge P(y) \wedge P(z) \to R(x,y,z) \vee \neg S(z,z,x,y))$$

$$\forall x(P(x) \to \exists yz(\neg R(y,z,x) \wedge (\neg R(x,y,z) \vee P(y))))$$

For interested readers we say that (non-restricted) *uniform one-dimensional fragment* is defined as above but in the last point of the definition the non-unary atoms must not necessarily use the whole set $\{x_0,\dots,x_k\}$ of variables but rather all those atoms use the same subset of this set (see [13]).

By AUF_1 we denote the extension of sUF_1 without equality with *alternation of quantifiers in blocks*. The set of σ-formulas of AUF_1 is the smallest set $\mathscr{F}$ such that:

- every σ-atom using at most one variable is in $\mathscr{F}$
- $\mathscr{F}$ is closed under Boolean connectives
- if $\varphi(x_0,\dots,x_k)$ is a Boolean combination of formulas in $\mathscr{F}$ with free variables in $\{x_0,\dots,x_k\}$ and atoms built out of precisely all of the variables $x_0,\dots,x_k$ (in an arbitrary order, possibly with repetitions) then $\mathrm{Q}_0x_0\dots\mathrm{Q}_kx_k\varphi$ and $\mathrm{Q}_1x_1\dots\mathrm{Q}_kx_k\varphi$ are in $\mathscr{F}$, where each Q_i is one of $\exists,\forall$.

Finally, we define a subset AUF_1^- of AUF_1 by requiring that its formulas are written in negation normal form NNF (that is negation is used only in front of atomic formulas, and the only other Boolean connectives are $\vee$ and $\wedge$), and that every sequence of quantifiers in the last point of the definition either contains only universal quantifiers or the last quantifier Q_k is existential. In AUF_1^- we can write, e.g.:

$$\forall xy\exists z(\neg P(x) \wedge \neg P(y) \vee R(x,y,z))$$

$$\forall x(P(x) \vee \exists y\forall z\exists t S(x,y,z,t))$$

$$\forall xyz(R(x,y,z) \vee \exists t T(x,t) \wedge \exists t T(y,t) \wedge \exists t T(z,t))$$

Observe that AUF_1^- contains the whole sUF_1 and hence also FO^2. (For example the FO^2 sentence $\exists x\forall y\psi(x,y)$ belongs to sUF_1, since one may think that it has two blocks of quantifiers, both of length one, and indeed one of them is purely universal and the other ends with $\exists$.)

[1]Please note that those atoms need not to belong to $\mathscr{F}$.

2.4 Normal forms and basic decidability result

We introduce normal form for AUF_1^- formulas, generalizing Scott's normal form for FO^2 (cf. [34, 11]). We start with a version involving 0-ary predicates, called *weak* normal form, and then explain how to remove them. In our normal form as well as in some intermediate formulas we allow ourselves to use implications which are usually not allowed in NNF formulas, but here they are very natural (note that converting them to disjunctions using the basic law $p \to q \equiv \neg p \vee q$ will not affect the blocks of quantifiers). We say that a AUF_1^- sentence is in *weak normal form* if it is a conjunction of formulas having one of the following shapes.

$$\mathrm{Q}_1 x_1 \ldots \mathrm{Q}_k x_k \psi(x_1,\ldots,x_k), \tag{1}$$
$$E \to \mathrm{Q}_1 x_1 \ldots \mathrm{Q}_k x_k \psi(x_1,\ldots,x_k), \tag{2}$$

where

- $k \geqslant 0$ is a natural number
- the x_i are distinct variables
- ψ is a quantifier-free AUF_1^- formula (Boolean combination of atoms, each of them containing all the variables $x_1,\ldots,x_k$ or at most one of them)
- every Q_i is a quantifier (universal or existential) and either all the Q_i are universal (*universal conjunct*) or Q_k is existential (*existential conjunct*)
- E is a 0-ary relation symbol

In particular, in a formula of type (1), k may be equal to 0; in this case ψ is a Boolean combination of 0-ary predicates.

Lemma 1. *Let φ be a AUF_1^- sentence. Then there exists a polynomially computable AUF_1^- sentence φ' in weak normal form over a signature extending the signature of φ by some fresh unary and 0-ary relation symbols, such that (i) every model of φ can be expanded to a model of φ' and (ii) every model of φ' is a model of φ.*

Proof. (Sketch) Assume that φ is in NNF. Take an innermost subformula ψ_0 starting with a maximal block of quantifiers. If it has a free variable, that is, is of the form

$$\mathrm{Q}_1 x_1 \ldots \mathrm{Q}_k x_k \psi(x_1,\ldots,x_k,y)$$

replace it by $P(y)$, for a fresh unary symbol P, and add the following normal form conjunct φ_{ψ_0} (partially) axiomatising P.

$$\forall y \mathrm{Q}_1 x_1 \ldots \mathrm{Q}_k x_k (P(y) \to \psi(x_1,\ldots,x_k,y)).$$

In other words, φ is replaced by $\varphi(P(y)/\psi_0) \wedge \varphi_{\psi_0}$.

If ψ_0 is a proper subsentence, that is it is of the form

$$\mathrm{Q}_1 x_1 \ldots \mathrm{Q}_k x_k \psi(x_1,\ldots,x_k),$$

then replace it by E, for a fresh 0-ary symbol E and add the conjunct

$$E \to \mathrm{Q}_1 x_1 \ldots \mathrm{Q}_k x_k \psi(x_1,\ldots,x_k).$$

Repeat this process as long as possible. Note that we indeed append conjuncts belonging to AUF_1^-.

The above process is similar to Scott's reduction of FO^2 formulas to their normal form. Besides the natural modifications needed to deal with sequences of quantifiers rather than with single quantifiers, the main difference is that in the appended conjuncts axiomatizing the freshly introduced unary and 0-ary predicates, we write implications in only one direction. This is sound as our initial formula is assumed to be in NNF. Indeed, consider a single step of the reduction, assuming that the case with a free variable in ψ_0 applies. In this step $\varphi' = \varphi(P(y)/\psi_0) \wedge \varphi_{\psi_0}$ is produced from φ. Assuming that $\mathfrak{A} \models \varphi$ we obtain a model $\mathfrak{A}'$ of φ' by making the unary relation P true at all elements a of A such that $\mathfrak{A} \models \psi_0[a]$. This makes the appended conjunct φ_{ψ_0} true; obviously, also $\varphi(P(y)/\psi_0)$ remains true. In the opposite direction assume $\mathfrak{A}' \models \varphi'$. It may happen that the subformula $\psi_0(y)$ of φ is true in $\mathfrak{A}'$ in more points than $P(y)$ is. However, to guarantee that φ is true it suffices that it is true *at least* at those points where $P(y)$ is, which is ensured by the appended conjunct φ_{ψ_0}; this is because φ is assumed to be in NNF and thus ψ_0 appears in φ in the scope of no negation symbol. We reason similarly for the case when ψ_0 is a subsentence. □

For our purposes, that is showing the finite model property for AUF$_1^-$ and demonstrating that its satisfiability problem is in NExpTime, we can further simplify our formulas, by eliminating 0-ary predicates. What we can do is to guess the truth values for all the 0-ary predicates and replace them by $\top$ or $\bot$, in accordance with the guess. In particular, the conjuncts of the form $E \rightarrow Q_1x_1 \ldots Q_kx_k\psi(x_1,\ldots,x_k)$ are eliminated if E is guessed to be $\bot$ and replaced just by $Q_1x_1 \ldots Q_kx_k\psi(x_1,\ldots,x_k)$ if E is guessed to be $\top$.

It is convenient to split the set of the resulting conjuncts into those whose all quantifiers are universal and those which end with the existential quantifier. We say that a sentence φ is in *normal form* if it is of the following shape:

$$\bigwedge_{1\leqslant i\leqslant m_\exists} \varphi_i^\exists \wedge \bigwedge_{1\leqslant i\leqslant m_\forall} \varphi_i^\forall, \tag{3}$$

where $\varphi_i^\exists = Q_1^i x_1 Q_2^i x_2 \ldots Q_{k_i-1}^i x_{k_i-1} \exists x_{k_i} \psi_i^\exists$, $\varphi_i^\forall = \forall x_1 \ldots x_{l_i} \psi_i^\forall$, for $Q_j^i \in \{\forall, \exists\}$, $\psi_i^\exists = \psi_i^\exists(x_1,x_2,\ldots,x_{k_i})$ and $\psi_i^\forall = \psi_i^\forall(x_1,\ldots,x_{l_i})$.

The discussion above justifies the following.

Lemma 2. *(i) The satisfiability problem for* AUF$_1^-$ *can be reduced in nondeterministic polynomial time to the satisfiability problem for normal form* AUF$_1^-$ *sentences. (ii) If the class of all normal form* AUF$_1^-$ *sentences has the finite (exponential) model property then also the whole* AUF$_1^-$ *has the finite (exponential) model property.*

The reduction to normal form described above allows us to easily prove the decidability of the satisfiability problem for AUF$_1^-$. This can be done by using the results on the Maslov Class $\overline{\mathrm{K}}$ (which is a dual of the Maslov class K). Full definition of $\overline{\mathrm{K}}$ is quite complicated and can be found, e.g., in [14]. For our purposes it is sufficient to say that when converted to prenex form $\overline{\mathrm{K}}$ formulas look as follows:

$$\exists y_1 \ldots \exists y_m \forall x_1 \ldots \forall x_k Q_1 z_1 \ldots Q_l z_l \psi, \tag{4}$$

where the Q_i are quantifiers, ψ is a quantifier-free formula without equality and every atom of ψ satisfies one of the following conditions: (i) it contains at most one x_i- or z_i-variable, (ii) it contains all the x_i-variables and no z_i-variables, or (iii) it contains an existentially quantified variable z_j and no z_i-variables with $i > j$.

Now, one easily observes that every AUF_1^--normal form conjunct belongs to $\overline{\mathrm{K}}$. Indeed, every $\varphi_i^{\forall}$-conjunct is of the form (4) with $m = l = 0$ and its every atom satisfies either condition (i) or (ii); every $\varphi_i^{\exists}$-conjunct is of the form (4) with $m = k = 0$ and every atom satisfying (i) or (iii).

Hence any normal form formula belongs to $\overline{\mathrm{DK}}$, the class of conjunctions of formulas in $\overline{\mathrm{K}}$. The satisfiability problem for $\overline{\mathrm{K}}$ was shown to be decidable in [25]. This result was extended to the class $\overline{\mathrm{DK}}$ in [14]. This gives us the basic decidability result.

Theorem 3. *The satisfiability problem for* AUF_1^- *is decidable.*

We recall that the precise complexity of $\overline{\mathrm{DK}}$ has not been established. It is also not known if $\overline{\mathrm{DK}}$ has the finite model property and if its finite satisfiability is decidable. The same questions for $\overline{\mathrm{K}}$ are also open.

3 Finite model property

The following theorem is the main results of this paper. Besides just proving the finite model property for AUF_1^-, it will also allow us to establish the exact complexity of its satisfiability problem.

Theorem 4. AUF_1^- *has the exponential model property.*

The rest of this section is devoted to a proof of the above theorem. By Lemma 2 we may restrict attention to formulas of the form (3).

3.1 Satisfaction forests

In this subsection we introduce *satisfaction forests*, which are auxiliary structures (partially) describing some finite models of normal form AUF_1^- sentences. We first explain how to extract a satisfaction forest from a given finite model $\mathfrak{B}$ of a normal form sentence φ. Then we formally define satisfaction forests and relate their existence to the existence of finite models of normal form sentences.

3.1.1 Extracting a satisfaction forest from a model

Let φ be a normal form AUF_1^- sentence and let $\mathfrak{B}$ be its finite model. Assume that φ is as in (3). The satisfaction forest will be a collection of labelled trees, one tree for each existential conjunct of φ, showing how this conjunct is satisfied in $\mathfrak{B}$. The labelling function will be denoted $\mathscr{L}$ and will assign elements from B to tree nodes (with the exception of the root, which will be assigned the special empty label).

Consider a single existential conjunct $\varphi_i^{\exists} = \mathrm{Q}_1^i x_1 \mathrm{Q}_2^i x_2 \ldots \mathrm{Q}_{k_i-1}^i x_{k_i-1} \exists x_{k_i} \psi_i^{\exists}$. Its satisfaction tree $\mathscr{T}_i$ is built in the following process.

Start with a root labelled with the empty label. The root forms level 0 of the tree. Level j, $0 < j \leqslant k_i$ will correspond to the quantifier Q_j^i. Assume level $j-1$ has been constructed, for $0 < j \leqslant k_i$. For each of its nodes n:

- If $\mathrm{Q}_j^i = \forall$ then for each element $b \in B$ add a child n' of n to $\mathscr{T}_i$ and set $\mathscr{L}(n') := b$. Nodes added in this step are called *universal nodes*.
- If $\mathrm{Q}_j^i = \exists$ then let $n_1, \ldots, n_{j-1} = n$ be the sequence of non-root nodes on the branch of n, ordered from the child of the root towards n. Choose in $\mathfrak{B}$ an element b such that

$$\mathfrak{B} \models \mathrm{Q}_{j+1}^i x_{j+1} \ldots \mathrm{Q}_{k_i-1}^i x_{k_i-1} \exists x_{k_i} \psi_i^{\exists}(\mathscr{L}(n_1), \ldots, \mathscr{L}(n_{j-1}), b, x_{k+1}, \ldots, x_{k_i}).$$

It is clear that such an element exists. If $j < k_i$ then we call it an *intermediate witness* for $\varphi_i^\exists$, and if $j = k_i$ we call it the *final witness* for $\varphi_i^\exists$. Add a single child n' of n to $\mathscr{T}_i$ and set $\mathscr{L}(n') = b$. The added element is called an *existential node*.

For a branch $\flat$ of the above-defined tree we denote by $Set(\flat)$ the set of labels of the non-root elements of $\flat$, by $Set^-(\flat)$ the set of labels of non-root and non-leaf elements of $\flat$ and by $Seq(\flat)$ the sequence of the non-root elements of $\flat$, ordered from the child of the root towards the leaf.

We further overload the function $\mathscr{L}$ by allowing it to define also labels for branches of the tree (by a *branch* we mean here a sequence of elements $n_1, \ldots, n_{k_i}$ such that n_1 is a child of the root, n_{k_i} is a leaf, and each n_{i+1} is a child of n_i). We label each branch $\flat$ with the pre-substructure of $\mathfrak{B}$ over $Set(\flat)$.

To declare some properties of satisfaction forests we need the following notions. A pre-structure $\mathfrak{H}^*$ is $\varphi_i^\forall$*-compatible*, if for every sequence $a_1, \ldots, a_{l_i}$ of elements of H such that $\{a_1, \ldots, a_{l_i}\} = H$ we have $\mathfrak{H}^* \models \psi_i^\forall(a_1, \ldots, a_{l_i})$. A pre-structure is $\varphi^\forall$*-compatible* if it is $\varphi_i^\forall$-compatible for every conjunct $\varphi_i^\forall$. Further, a set of 1-types $\{\alpha_1, \ldots, \alpha_k\}$ is $\varphi^\forall$*-compatible* if for a set of distinct elements $H = \{a_1, \ldots, a_m\}$ and any assignment $f : \{a_1, \ldots, a_m\} \to \{\alpha_1, \ldots, \alpha_k\}$ one can build a $\varphi^\forall$-compatible pre-structure on H in which, for every i, the 1-type of a_i is $f(a_i)$.

We now collect some properties of the tree $\mathscr{T}_i$ for $\varphi_i^\exists$ constructed as above.

(T1) for $1 \leqslant j \leqslant k_i$, and every node n from level $j-1$:
  (a) if $Q_j^i = \forall$ then n has precisely $|B|$ children, labelled by distinct elements of B (recall that each of these children is called a universal node)
  (b) if $Q_j^i = \exists$ then n has precisely one child (recall that this child is called an existential node)

(T2) for every branch $\flat \in \mathscr{T}_i$, assuming $Seq(\flat) = (a_1, \ldots, a_{k_i})$, we have $\mathscr{L}(\flat) \models \psi_i^\exists(a_1, \ldots, a_{k_i})$

(T3) for every pair of branches $\flat_1, \flat_2 \in \mathscr{T}_i$, for every $a \in B$ such that $a \in Set(\flat_1)$ and $a \in Set(\flat_2)$ the 1-types of a in $\mathscr{L}(\flat_1)$ and in $\mathscr{L}(\flat_2)$ are identical

(T4) for every pair of branches $\flat_1, \flat_2 \in \mathscr{T}_i$ such that $Set(\flat_1) = Set(\flat_2)$ we have that $\mathscr{L}(\flat_1) = \mathscr{L}(\flat_2)$.

(T5) for every branch $\flat \in \mathscr{T}_i$, $\mathscr{L}(\flat)$ is $\varphi^\forall$-compatible

Now we collect some properties of the whole sequence of trees $\mathscr{T}_1, \ldots, \mathscr{T}_{m_\exists}$ constructed for φ and $\mathfrak{B}$.

(F1) for every i, $\mathscr{T}_i$ is a satisfaction tree over B for $\varphi_i^\exists$

(F2) for every pair of branches $\flat_1 \in \mathscr{T}_i, \flat_2 \in \mathscr{T}_j$, $i \neq j$, for every $a \in B$ such that $a \in Set(\flat_1)$ and $a \in Set(\flat_2)$ the 1-types of a in $\mathscr{L}(\flat_1)$ and in $\mathscr{L}(\flat_2)$ are identical

(F3) for every pair of branches $\flat_1 \in \mathscr{T}_i, \flat_2 \in \mathscr{T}_j$, $i \neq j$ such that $Set(\flat_1) = Set(\flat_2)$ we have that $\mathscr{L}(\flat_1) = \mathscr{L}(\flat_2)$

(F4) the set of all 1-types appearing in the pre-structures defined as labels of the branches of the trees in $\mathscr{F}$ is $\varphi^\forall$-compatible.

Properties (T3), (T4), (F2) and (F3) will be sometimes called the *(forest) consistency conditions*.

Claim 5. *The sequence of trees $\mathscr{T}_1, \ldots, \mathscr{T}_{m_\exists}$ constructed as above for the structure $\mathfrak{B}$ and the sentence φ satisfies conditions (T1)-(T5) and (F1)-(F4).*

Proof. (Sketch) It is not difficult to see that each of the $\mathscr{T}_i$ satisfies (T1)-(T5) and that the whole sequence satisfies (F1)-(F3). The only non-obvious point is (F4). Let us prove that it is true. Let $\alpha_1, \ldots, \alpha_k$ be the list of all 1-types appearing in the pre-structures defined in the whole forest. Let $H = \{a_1, \ldots, a_m\}$ be a set of fresh distinct elements and $f : \{a_1, \ldots, a_m\} \to \{\alpha_1, \ldots, \alpha_k\}$ an assignment of 1-types to these elements. We need to construct a $\varphi^{\forall}$-compatible pre-structure on H in which, for every i, the 1-type of a_i is $f(a_i)$. For each i choose an element $g(a_i) \in B$ such that $\text{tp}^{\mathfrak{B}}(g(a_i)) = f(a_i)$; g need not be injective. Let us define the pre-structure $\mathfrak{H}^*$ on H by setting the 1-type of a_i to be $f(a_i)$, and for every relation symbol R, and every sequence $c_1, \ldots, c_l$ of elements of H such that l is the arity of R and $\{c_1, \ldots, c_l\} = H$, setting the truth-value of the atom $R(c_1, \ldots, c_l)$ to be equal to the truth-value of $R(g(c_1), \ldots, g(c_l))$ in $\mathfrak{B}$. We claim that so defined $\mathfrak{H}^*$ is $\varphi^{\forall}$-compatible. To see this take any conjunct $\varphi_i^{\forall}$ and any sequence of elements $c_1, \ldots, c_{l_i}$ such that $\{c_1, \ldots, c_{l_i}\} = H$ and assume to the contrary that $\mathfrak{H}^* \not\models \psi_i^{\forall}(c_1, \ldots, c_{l_i})$. But then $\mathfrak{B} \not\models \psi_i^{\forall}(g(c_1), \ldots, g(c_{l_i}))$, as the truth-values of the atoms appearing in $\psi_i^{\forall}$ in the two considered structures appropriately coincide by our definition of $\mathfrak{H}^*$. Contradiction. □

3.1.2 Satisfaction forests and the existence of finite models

Let φ be a normal form AUF_1^- sentence (we do not assume that a model of φ is known). Formally, a *satisfaction forest* for φ *over a domain* B is a sequence of trees $\mathscr{T}_1, \ldots, \mathscr{T}_{m_\exists}$ together with a labelling function $\mathscr{L}$, assigning elements of B to the nodes of the $\mathscr{T}_i$ (with the exception of their roots to which the special empty label is assigned) and pre-structures to their branches, such that each of the trees $\mathscr{T}_i$ satisfies conditions (T1)-(T5) and the whole sequence satisfies conditions (F1)-(F4).

Lemma 6. *A normal form* AUF_1^- *sentence* φ *has a finite model over a domain* B *iff it has a satisfaction forest over* B.

Proof. Left-to-right implication is justified by the extraction of a satisfaction forest from a given finite model of φ described in Section 3.1.1, and in particular by Claim 5.

In the opposite direction assume that a satisfaction forest over a finite domain B for φ is given. We construct a model $\mathfrak{B}$ of φ over the domain B. The construction is natural:

Step 1: 1-types. The 1-type of an element $b \in B$ is defined as the 1-type of b in the structure $\mathscr{L}(\flat)$ for an arbitrarily chosen branch $\flat$, in an arbitrarily chosen tree $\mathscr{T}_i$, for which $b \in \mathit{Set}(\flat)$.

Step 2: Witnesses. For every tree $\mathscr{T}_i$ and its every branch $\flat$ of $\mathscr{T}_i$ define the pre-structure on $\mathit{Set}(\flat)$ in accordance with $\mathscr{L}(\flat)$.

Step 3: Completion. For any set of distinct elements $\{b_1, \ldots, b_k\}$ whose pre-structure is not yet defined, choose any $\varphi^{\forall}$-compatible pre-structure which retains the already defined 1-types of the b_i.

Properties (T3), (F2), (T4) and (F3) guarantee that Step 1 and Step 2 can be performed without conflicts and the existence of an appropriate pre-structure in Step 3 is guaranteed by (F4).

It remains to see that $\mathfrak{B} \models \varphi$. Consider any existential conjunct of φ, that is a conjunct $\varphi_i^{\exists} = \mathrm{Q}_1^i x_1 \mathrm{Q}_2^i x_2 \ldots \mathrm{Q}_{k_i-1}^i x_{k_i-1} \exists x_{k_i} \psi_i^{\exists}$. The satisfaction tree $\mathscr{T}_i$ witnesses that $\varphi_i^{\exists}$ indeed holds: it describes all possible substitutions for universally quantified variables, and shows how intermediate and final witnesses for existential quantifiers can be chosen. Consider now any universal conjunct $\varphi_i^{\forall} = \forall x_1 \ldots x_{l_i} \psi_i^{\forall}$ and let $b_1, \ldots, b_{l_i}$ be any sequence of elements of B (possibly with repetitions). Let $H = \{b_1, \ldots, b_{l_i}\}$. The pre-structure on H has been defined either in Step 2 or in Step 3. In both cases we know that it is $\varphi^{\forall}$-compatible, in particular it is $\varphi_i^{\forall}$-compatible, so $\mathfrak{H}^* \models \psi_i^{\forall}(b_1, \ldots, b_{l_i})$. □

3.2 From a model to a satisfaction forest over a small domain

We are ready to present the main construction of this paper in which we show that every satisfiable formula has a satisfaction forest over a small domain.

Let $\mathfrak{A}$ be a (possibly infinite) model of a normal form sentence φ of the shape as in (3). We show how to construct a satisfaction forest over a domain of size bounded exponentially in $|\varphi|$. By Lemma 6 this will guarantee that φ has a finite model over such a bounded domain.

3.2.1 Domain

Let L be the number of 1-types (over the signature of φ) realized in $\mathfrak{A}$, and let these types be enumerated as $\alpha_1,\ldots,\alpha_L$. Let $K=\max\{k_i : 1\leqslant i\leqslant m_\exists\}$. We define the domain B to be $\{1,\ldots,2K\}\times\{1,\ldots,m_\exists\}\times\{1,\ldots,(K-1)^{K-1}\}\times\{1,\ldots,L\}$. Note that K and $m_\exists$ are bounded linearly and L is bounded exponentially in $|\varphi|$, and hence $|B|$ is indeed bounded exponentially in $|\varphi|$.

For convenience let us split B into the sets $B_i=\{(i,*,*,*)\}$ (here and in the sequel $*$ will be sometimes used as a wildcard in the tuples denoting elements of the domain). We will sometimes call B_i the i-th *layer* of B.

3.2.2 Some simple combinatorics: Extension functions

During the construction of the satisfaction forest we will design a special strategy for assigning labels to the leaves. To this end we introduce an auxiliary combinatorial tool, which we will call *extension functions*.

Let us recall a well known Hall's marriage theorem. A *matching* in a bipartite graph (G_1,G_2,E) is a partial injective function $f: G_1\to G_2$ such that if $f(a)=b$ then $(a,b)\in E$.

Theorem 7 (Hall). *Let (G_1,G_2,E) be a bipartite graph. There exists a matching covering G_1 iff for any set $W\subseteq G_1$ the number of vertices of G_2 incident to the edges emitted from W is greater or equal to $|W|$.*

For a natural number n, let $[n]$ denote the set $\{1,\ldots,n\}$ and for $1\leqslant l\leqslant n$ let $[n]^l$ denote the set of all subsets of $[n]$ of cardinality l.

Lemma 8. *For every $0<l<K$ there exists a $1{-}1$ function $ext_l : [2K]^l\to[2K]^{l+1}$ such that for any $S\in[2K]^l$ we have that $S\subseteq ext_l(S)$.*

Proof. Consider the bipartite graph $([2K]^l,[2K]^{l+1},E)$ such that $(S,S')\in E$ iff $S\subseteq S'$. To show that a desired ext_l exists it suffices to show the existence of a matching covering entirely the set $[2K]^l$. To this end we apply Hall's marriage theorem. In our graph every node from $[2K]^l$ has degree $2K-l$ (given an l-element subset of $[2K]$ it can be expanded to an $l+1$ subset just by adding to it precisely one of the remaining $2K-l$ elements) and every node from $[2K]^{l+1}$ has degree $l+1$ (to obtain an l-element subset of a $l+1$-subset one just removes one of the elements of the latter). Take a subset W of $[2K]^l$. The nodes of this subset are incident to $|W|\cdot(2K-l)$ edges in total. Let us see that the number of nodes in $[2K]^{l+1}$ incident to a node from W is greater than or equal to $|W|$. Indeed, assume to the contrary that it is not. Then at most $|W|-1$ nodes absorb $|W|\cdot(2K-l)$ edges emitted by W, but this means that $|W|\cdot(2K-l)\leqslant(|W|-1)\cdot(l+1)$. Rearranging this inequality we get that $|W|(2K-2l-1)+l+1\leqslant 0$. But using the assumption that $0<l<K$ we have that $(2K-2l-1)>0$ and hence the whole left-hand side of the last inequality must be greater than 0. Contradiction. Thus our graph satisfies the Hall's theorem assumptions which guarantee the existence of a matching from $[2K]^l$ to $[2K]^{l+1}$, covering entirely $[2K]^l$. This matching can be taken as ext_l. □

Choose an extension function ext_l for every l and let $ext=\bigcup_{l=1}^{K-1} ext_l$, that is, ext is a function which takes a non-empty subset of $[2K]$ of size at most $K-1$ and returns a superset containing precisely one new element. Obviously ext remains an injective function.

3.2.3 Construction of a satisfaction forest

We now describe how to construct a satisfaction forest $\mathscr{T}_1, \ldots, \mathscr{T}_{m_\exists}$ for φ over the domain B. It should be helpful to announce how we are going to take care of the consistency conditions for the whole forest:

- Conditions (F2) and (T3): With every element $a = (*,*,*,l) \in B$ we associate the 1-type α_l. Whenever a will be used as a label of a node in a satisfaction tree then its 1-type in the pre-structure defined for any branch containing a node labelled with a will be set to α_l.
- Conditions (F3) and (T4): for a a pair of distinct branches $\flat_1$, $\flat_2$ (either belonging to the same tree or to two different trees) we will simply have $Set(\flat_1) \neq Set(\flat_2)$. This condition will be ensured by an appropriate use of the extension function. Here it is important that the last quantifier in every $\varphi_i^\exists$-conjunct is existential, and hence the last node of every branch in $\mathscr{T}_i$ is also existential, so we can freely choose its label from B.

Let us explain how to construct a single $\mathscr{T}_i$, a $\varphi_i^\exists$-satisfaction tree over B. The general shape of $\mathscr{T}_i$ is determined by $\varphi_i^\exists$ and B: we know how many nodes we need, we know which of them are existential, and which are universal, we know the labels of the universal nodes. It remains to assign labels to existential nodes (elements of B) and to branches (pre-structures on the set of elements formed from the labels of the nodes on a branch).

We define an auxiliary function *pat* which for every node of $\mathscr{T}_i$ returns *a pattern element* from $\mathfrak{A}$. We will choose $pat(n)$, so that its 1-type is equal to type of $\mathscr{L}(n)$. We remark, that if two nodes from different branches have the same label then they do not need to have the same pattern element.

Consider a node n_k and assume that all its non-root ancestors $n_1, \ldots, n_{k-1}$ have the function *pat* and their labels already defined. We proceed as follows

- If n_k is universal then its label $\mathscr{L}(n_k)$ is known
  - If $\mathscr{L}(n_k) = \mathscr{L}(n_j)$ for some $j < k$ then we set $pat(n_k) = pat(n_j)$.
  - If the label $\mathscr{L}(n_k)$ is not used by the ancestors of n then choose as $pat(n)$ an arbitrary element of $\mathfrak{A}$ of the 1-type assigned to $\mathscr{L}(n_k)$. (In particular, we may use an element which was used by one of the ancestors of n)
- If n_k is existential then we need to define both $\mathscr{L}(n)$ and $pat(n)$. By our construction we have that

  $$\mathfrak{A} \models \exists x_k \mathrm{Q}^i_{k+1} x_{k+1} \ldots \mathrm{Q}^i_{k_i-1} x_{k_i-1} \exists x_{k_i} \psi_i^\exists(pat(n_1), \ldots, pat(n_{k-1}), x_k, x_{k+1}, \ldots, x_{k_i}).$$

  We choose an element $w \in A$ witnessing the previous formula, i.e., an element such that

  $$\mathfrak{A} \models \mathrm{Q}^i_{k+1} x_{k+1} \ldots \mathrm{Q}^i_{k_i-1} x_{k_i-1} \exists x_{k_i} \psi_i^\exists(pat(n_1), \ldots, pat(n_{k-1}), w, x_{k+1}, \ldots, x_{k_i})$$

  and set $pat(n_k) = w$. To define the label of n_k we consider two cases:

  - If n_k is not a leaf then:
    - if $pat(n_j) = w$ for some $j < k$ then set $\mathscr{L}(n_k) = \mathscr{L}(n_j)$
    - otherwise we choose as $\mathscr{L}(n_k)$ an arbitrary element of B which has assigned the 1-type $\mathrm{tp}^{\mathfrak{A}}(w)$, not used by the ancestors of n_k (there are many copies of each 1-type in B so it is always possible).
  - If n_k is a leaf then let $\flat$ be the branch of n_k and let $S = \{j : n_l \in B_j \text{ for some } l < k\}$. Of course, $|S| < k \leqslant K$ so $ext(S)$ is defined. Let s be the unique member of $ext(S) \setminus S$. We take as $\mathscr{L}(n_k)$ an element $(s,i,t,l) \in B_s$ where l is such that $\alpha_l = \mathrm{tp}^{\mathfrak{A}}(w)$, and where t is chosen so that none

of the branches $\flat'$ of the current tree for which the labels have been already defined such that $Set^-(\flat') = Set^-(\flat)$ used (s,i,t,l) as the label of its leaf. We indeed have enough elements for this, since obviously $|Set^-(\flat)| \leqslant K-1$ and thus there are at most $(K-1)^{K-1}$ different branches whose nodes from the first $K-1$ levels are labelled by elements of $|Set^-(\flat)|$ (recall that there are $(K-1)^{K-1}$ possible choices for t).

Take now any branch $\flat$ of $\mathscr{T}_i$. It remains to define the pre-structure $\mathscr{L}(\flat)$. For any relational symbol R of arity m and any sequence $a_{i_1},\ldots,a_{i_m}$ of elements of $Set(\flat)$ containing all the elements of $Set(\flat)$ we set $R(a_{i_1},\ldots,a_{i_m})$ to be true iff $R(pat(a_{i_1}),\ldots,pat(a_{i_m}))$ is true in $\mathfrak{A}$. For every a_j its 1-type is set to be equal to the 1-type of $pat(a_j)$. This completes the definition of the pre-structure on $Set(\flat)$. Note that this ensures that this pre-structure satisfies $\psi_i^{\exists}(Seq(\flat))$.

3.3 Correctness

Let us now see that the defined satisfaction forest indeed satisfies all the required conditions.

- Conditions (T1), (T2) and (T3) should be clear.
- For (T4) we show that there is no pair of branches $\flat$, $\flat'$ in a tree $\mathscr{T}_i$ with $Set(\flat) = Set(\flat')$. Indeed, we have chosen as labels of the leaves of $\flat$ and $\flat'$ two different elements $b = (s,i,x,*)$ and $b' = (s,i,y,*)$ of a layer B_s which is not inhabited by the elements of $Set^-(\flat)$ or $Set^-(\flat')$ (due to the use of the function *ext*). So $b \in Set(\flat)$ but $b \notin Set(\flat')$ and thus $Set(\flat) \neq Set(\flat')$.
- To show that (T5) holds assume to the contrary that for some branch $\flat$, $\mathfrak{H}^* = \mathscr{L}(\flat)$ is not $\varphi^{\forall}$-compatible; take i for which it is not $\varphi_i^{\forall}$-compatible. So, for some sequence $a_1,\ldots,a_{l_1}$ such that $\{a_1,\ldots,a_{l_1}\} = Set(\flat) = H$ we have $\mathfrak{H}^* \not\models \psi_i^{\forall}(a_1,\ldots,a_{l_i})$. But then the definition of the pre-structure in $\mathscr{L}(\flat)$ implies that $\mathfrak{A} \not\models \psi_i^{\forall}(pat(a_1),\ldots,pat(a_{l_i}))$, that is $\mathfrak{A}$ violates $\varphi_i^{\forall}$. Contradiction.
- Conditions (F1), (F2) should be clear.
- For (F3) the argument is similar to the argument for (T4): We show that there is is pair of branches $\flat_1$, $\flat_2$ in a tree $\mathscr{T}_i$, and resp., $\mathscr{T}_j$, $i \neq j$, with $Set(\flat_1) = Set(\flat_2)$. Again, this follows from the fact that we have chosen as labels of the leaves of $\flat_1$ and $\flat_2$ two different elements b and b' of a layer B_s which is not inhabited by the elements of $Set^-(\flat_1)$ or $Set^-(\flat_2)$. This time the elements b_1 and b_2 are different from each other since $b_1 = (s,i,*,*)$ and $b_2 = (s,j,*,*)$.
- For (F4) we reason precisely as in the reasoning for (F4) in the proof of the Claim in Section 3.1 (we just replace the structure $\mathfrak{B}$ from this proof with the currently considered structure $\mathfrak{A}$).

An immediate consequence of Thm. 4 is:

Theorem 9. *The satisfiability problem for* AUF$_1^-$ *is* NEXPTIME*-complete.*

Proof. The lower bound is inherited from the lower bound for FO^2 [24]. Let us turn to the upper bound.

By Lemma 2 it suffices to show how to decide satisfiability of a normal form sentence φ. By Theorem 4 if φ is satisfiable then it has a model with exponentially bounded domain. We guess some natural description of such a model $\mathfrak{A}$. We note that this description is also of exponential size with respect to $|\varphi|$: Indeed, we need to describe some number (linearly bounded in $|\varphi|$) of relations of arity at most $|\varphi|$, and it is straightforward, taking into consideration the size of the domain, that a description of a single such relation is at most exponential in $|\varphi|$. A verification of a single normal form conjunct in the guessed structure can be done in an exhaustive way, by considering all possible substitutions for the variables.

Alternatively, instead of guessing a model one could guess a satisfaction forest for φ. Again, a routine inspection reveals that the size of its description can be bounded exponentially in $|\varphi|$; also the verification of the properties (T1)-(T5), (F1)-(F4) would not be problematic. $\square$

4 Infinity axiom with free use of equality

In this section we note that allowing for free use of equality in our logic changes the situation significantly: we lose the finite model property. We recall that in the case of UF_1 free use of equality does not spoil the decidability and even does not change the complexity.

In the recent paper [10] we note that the fragment with arbitrary blocks of quantifiers AUF_1 and with free use of equality contains infinity axioms (satisfiable formulas without finite models), by constructing the following three-variable formula:

$$\exists x S(x) \wedge \forall x \exists y \forall z (\neg S(y) \wedge R(x,y,z) \wedge (x = z \vee \neg R(z,y,x))),$$

which has no finite models but is satisfied in the model whose universe is the set of natural numbers, S is true only at 0 and $Rxyz$ is true iff $y = x+1$.

The above example can be simply adapted to the case of AUF_1^- with free use of equality. We just add a dummy existentially quantified variable t and require it to be equal to the previous, universally quantified variable z. To accommodate all the variables we increase the arity of R by 2 (one can think that the first and the last position of R from the previous example have been doubled):

$$\exists x S(x) \wedge \forall x \exists y \forall z \exists t.(t = z \wedge \neg S(y) \wedge R(x,x,y,z,t) \wedge (x = z \vee \neg R(z,t,y,x,x))).$$

5 Conclusions

We identified a non-trivial uniform one-dimensional logic in which a use of mixed blocks of quantifiers is allowed, strictly extending the two-variable fragment FO^2 without equality and the previously defined fragment sUF_1 without equality. We proved that, similarly to FO^2 and sUF_1, this logic has the finite, exponential model property and NExpTime-complete satisfiability problem.

There are two interesting directions, orthogonal to each other, in which it would be valuable to extend our work. The first is investigating the decidability, complexity and the status of the finite model property for full AUF_1 without equality, that is to see what happens to our logic if arbitrary blocks of quantifiers, possibly ending with the universal quantifier, are allowed. As already mentioned, in our recent work [10] we answered this question for the three variable restriction, AUF_1^3, of AUF_1 by showing the exponential model property and NExpTime-completeness of its satisfiability problem.

The second idea is to revive the research on Maslov Class $\overline{\mathrm{K}}$, by attempting to determine the precise complexity of its satisfiability problem and investigating whether it has the finite model property. When designing the fragment AUF_1^- we took some inspiration from the definition of $\overline{\mathrm{K}}$, and indeed we were able to reduce satisfiability of the former to the latter. We believe that what we have learned working on AUF_1^- will prove useful in the case of $\overline{\mathrm{K}}$.

There are also some probably slightly less attractive, but still interesting, a bit more technical questions that one can try to answer. For example, what happens to our logic if a use of equalities/inequalities (free or uniform) or constants is allowed.

Acknowledgement

This work is supported by NCN grant No. 2021/41/B/ ST6/00996.

References

[1] H. Andréka, J. van Benthem & I. Németi (1998): *Modal Languages and Bounded Fragments of Predicate Logic*. Journal of Philosophical Logic 27, pp. 217–274, doi:10.1023/A:1004275029985.

[2] V. Bárány, B. ten Cate & L. Segoufin (2015): *Guarded Negation*. J. ACM 62(3), p. 22, doi:10.1145/2701414.

[3] S. Benaim, M. Benedikt, W. Charatonik, E. Kieronski, R. Lenhardt, F. Mazowiecki & J. Worrell (2016): *Complexity of Two-Variable Logic on Finite Trees*. ACM Trans. Comput. Log. 17(4), pp. 32:1–32:38, doi:10.1145/2996796.

[4] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick & L. Segoufin (2011): *Two-variable logic on data words*. ACM Trans. Comput. Log. 12(4), p. 27, doi:10.1145/1970398.1970403.

[5] M. Bojanczyk, A. Muscholl, T. Schwentick & L. Segoufin (2009): *Two-variable logic on data trees and XML reasoning*. J. ACM 56(3), doi:10.1145/1516512.1516515.

[6] P. Bourhis, M. Morak & A. Pieris (2017): *Making Cross Products and Guarded Ontology Languages Compatible*. In: IJCAI 2017, pp. 880–886, doi:10.24963/ijcai.2017/122.

[7] B. ten Cate & L. Segoufin (2013): *Unary negation*. Logical Methods in Comp. Sc. 9(3), doi:10.2168/LMCS-9(3:25)2013.

[8] W. Charatonik & P. Witkowski (2013): *Two-Variable Logic with Counting and Trees*. In: LICS 2013, pp. 73–82, doi:10.1109/LICS.2013.12.

[9] W. Charatonik & P. Witkowski (2015): *Two-variable Logic with Counting and a Linear Order*. In: CSL 2015, LIPIcs 41, pp. 631–647, doi:10.4230/LIPIcs.CSL.2015.631.

[10] O. Fiuk & E. Kieroński (2023): *An excursion to the border of decidability: between two- and three-variable logic*. In: LPAR 2023, EPiC Series in Computing 94, pp. 205–223, doi:10.29007/1xns.

[11] E. Grädel, P. Kolaitis & M. Y. Vardi (1997): *On the decision problem for two-variable first-order logic*. Bulletin of Symbolic Logic 3(1), pp. 53–69, doi:10.2307/421196.

[12] E. Grädel, M. Otto & E. Rosen (1997): *Two-variable logic with counting is decidable*. In: LICS 1997, pp. 306–317, doi:10.1109/LICS.1997.614957.

[13] L. Hella & A. Kuusisto (2014): *One-dimensional Fragment of First-order Logic*. In: Proceedings of Advances in Modal Logic, 2014, pp. 274–293. Available at http://www.aiml.net/volumes/volume10/Hella-Kuusisto.pdf.

[14] U. Hustadt & R. Schmidt (1999): *Maslov's Class K Revisited*. In: Automated Deduction — CADE-16, pp. 172–186, doi:10.1007/3-540-48660-7_12.

[15] Y. Kazakov (2006): *Saturation-based decision procedures for extensions of the guarded fragment*. Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany.

[16] E. Kieronski & A. Kuusisto (2014): *Complexity and Expressivity of Uniform One-Dimensional Fragment with Equality*. In: MFCS 2014, Part I, Lecture Notes in Computer Science 8634, pp. 365–376, doi:10.1007/978-3-662-44522-8_31.

[17] E. Kieronski & A. Kuusisto (2015): *Uniform One-Dimensional Fragments with One Equivalence Relation*. In: CSL 2015, LIPIcs 41, pp. 597–615, doi:10.4230/LIPIcs.CSL.2015.597.

[18] E. Kieronski, J. Michaliszyn, I. Pratt-Hartmann & L. Tendera (2014): *Two-Variable First-Order Logic with Equivalence Closure*. SIAM J. Comput. 43(3), pp. 1012–1063, doi:10.1137/120900095.

[19] E. Kieroński & M. Otto (2012): *Small Substructures and Decidability Issues for First-Order Logic with Two Variables*. Journal of Symbolic Logic 77, pp. 729–765, doi:10.2178/jsl/1344862160.

[20] Emanuel Kieronski (2019): *One-Dimensional Guarded Fragments*. In: MFCS 2019, LIPIcs 138, pp. 16:1–16:14, doi:10.4230/LIPIcs.MFCS.2019.16.

[21] Emanuel Kieronski & Sebastian Rudolph (2021): *Finite Model Theory of the Triguarded Fragment and Related Logics*. In: LICS 2021, pp. 1–13, doi:10.1109/LICS52264.2021.9470734.

[22] Andreas Krebs, Kamal Lodaya, Paritosh K. Pandya & Howard Straubing (2020): *Two-variable logics with some betweenness relations: Expressiveness, satisfiability and membership*. Log. Methods Comput. Sci. 16(3), doi:10.23638/LMCS-16(3:16)2020.

[23] Antti Kuusisto (2016): *On the Uniform One-dimensional Fragment*. In: Description Logics 2016, CEUR Workshop Proceedings 1577.

[24] H. R. Lewis (1980): *Complexity results for classes of quantificational formulas*. Journal of Computer and System Sciences 21(3), pp. 317 – 353, doi:10.1016/0022-0000(80)90027-6.

[25] S. J. Maslov (1971): *The inverse method for establishing deducibility for logical calculi*. The Calculi of Symbolic Logic I: Proceedings of the Steklov Institute of Mathematics 98.

[26] M. Mortimer (1975): *On languages with two variables*. Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 21, pp. 135–140, doi:10.1002/malq.19750210118.

[27] L. Pacholski, W. Szwast & L. Tendera (1997): *Complexity of two-variable logic with counting*. In: LICS 1997, pp. 318–327, doi:10.1109/LICS.1997.614958.

[28] I. Pratt-Hartmann (2010): *The Two-Variable Fragment with Counting Revisited*. In: WoLLIC 2010, pp. 42–54, doi:10.1007/978-3-642-13824-9_4.

[29] Ian Pratt-Hartmann (2015): *The two-variable fragment with counting and equivalence*. Math. Log. Q. 61(6), pp. 474–515, doi:10.1002/malq.201400102.

[30] Ian Pratt-Hartmann (2021): *Fluted Logic with Counting*. In: ICALP 2021, LIPIcs 198, pp. 141:1–141:17, doi:10.4230/LIPIcs.ICALP.2021.141.

[31] Ian Pratt-Hartmann, Wieslaw Szwast & Lidia Tendera (2019): *The Fluted Fragment Revisited*. J. Symb. Log. 84(3), pp. 1020–1048, doi:10.1017/jsl.2019.33.

[32] W. V. Quine (1969): *On the limits of decision*. In: Proceedings of the 14th International Congress of Philosophy, III, pp. 57–62.

[33] Sebastian Rudolph & Mantas Šimkus (2018): *The Triguarded Fragment of First-Order Logic*. In: LPAR 2018, EPiC Series in Computing 57, pp. 604–619, doi:10.29007/m8ts.

[34] Dana Scott (1962): *A decision method for validity of sentences in two variables*. Journal Symbolic Logic 27, p. 477.

FMplex: A Novel Method for Solving Linear Real Arithmetic Problems

Jasper Nalbach[*]
RWTH Aachen University, Germany
nalbach@cs.rwth-aachen.de

Valentin Promies
RWTH Aachen University, Germany
promies@cs.rwth-aachen.de

Erika Ábrahám
RWTH Aachen University, Germany
abraham@cs.rwth-aachen.de

Paul Kobialka
University of Oslo, Norway
paulkob@ifi.uio.no

In this paper we introduce a novel quantifier elimination method for conjunctions of *linear real arithmetic* constraints. Our algorithm is based on the *Fourier-Motzkin variable elimination* procedure, but by case splitting we are able to reduce the worst-case complexity from doubly to singly exponential. The adaption of the procedure for SMT solving has strong correspondence to the *simplex algorithm*, therefore we name it *FMplex*. Besides the theoretical foundations, we provide an experimental evaluation in the context of SMT solving.

1 Introduction

Linear real arithmetic (LRA) is a powerful first-order theory with strong practical relevance. We focus on checking the satisfiability of *conjunctions* of LRA constraints, which is needed e.g. for solving quantifier-free LRA formulas using *satisfiability modulo theories (SMT) solvers*. The problem is known to be solvable in *polynomial* worst-case complexity but, surprisingly, the *ellipsoid* method [13] proposed in 1980 by Khachiyan is still the only available algorithm that implements this bound. However, this method is seldomly used in practice due to its high average-case effort. Instead, most approaches employ the *simplex* algorithm introduced by Dantzig in 1947, which has a *singly exponential* worst case complexity, but which is quite efficient in practice. A third available solution is the *Fourier-Motzkin variable elimination (FM)* method, proposed in 1827 by Fourier [9] and re-discovered in 1936 by Motzkin [23]. In contrast to the other two approaches, FM admits quantifier elimination, but it has a *doubly exponential* worst case complexity, even though there have been various efforts to improve its efficiency by recognizing and avoiding redundant computations (e.g. [11, 12]).

In this paper, we propose a novel method, which is derived from the FM method, but which turns out to have striking resemblance to the simplex algorithm. This yields interesting theoretical insights into the relation of the two established methods and the nature of the problem itself. More precisely, our contributions include:

- The presentation of *FMplex*, a new variable elimination method based on a divide-and-conquer approach. We show that it does not contain certain redundancies Fourier-Motzkin might generate and it lowers the overall complexity from *doubly* to *singly* exponential.
- An adaptation of FMplex for SMT solving, including methods to prune the search tree based on structural observations.

[*]Jasper Nalbach was supported by the DFG RTG 2236 UnRAVeL.

- A theorem formalizing connections between FMplex and the simplex algorithm.
- An implementation of the SMT adaptation and its experimental evaluation.

After recalling necessary preliminaries in Section 2, we introduce our novel FMplex method first for quantifier elimination in Section 3 and then for SMT solving in Section 4. We present related work and compare FMplex with other methods, first qualitatively in Section 5, and then experimentally in Section 6. We discuss future work and conclude the paper in Section 7.

An extended version of this paper including more detailed proofs can be found on arXiv [25].

2 Preliminaries

Let $\mathbb{R}$, $\mathbb{Q}$ and $\mathbb{N}$ denote the set of real, rational respectively natural ($0 \notin \mathbb{N}$) numbers. For $k \in \mathbb{N}$ we define $[k] := \{1,\dots,k\}$. Throughout this paper, we fix $n \in \mathbb{N}$, a set $X = \{x_1,\dots,x_n\}$ and a corresponding vector $\boldsymbol{x} = (x_1,\dots,x_n)^T$ of $\mathbb{R}$-valued variables.

Matrices For $m \in \mathbb{N}$ let $E^{(m)} \in \mathbb{Q}^{m\times m}$ be the identity matrix, and $\boldsymbol{0}^{(m)} = (0 \ \cdots \ 0)^T \in \mathbb{Q}^{m\times 1}$. The ith component of $\boldsymbol{f} \in \mathbb{Q}^{m\times 1} \cup \mathbb{Q}^{1\times m}$ is denoted by f_i and the component-wise comparison to zero by $\boldsymbol{f} \geq 0$. For $A \in \mathbb{Q}^{m\times n}$, $\boldsymbol{a}_{i,-} \in \mathbb{Q}^{1\times n}$ and $\boldsymbol{a}_{-,i} \in \mathbb{Q}^{m\times 1}$ denote the ith row respectively column vector of A. Furthermore, $A[I]$ denotes the sub-matrix of A containing only the rows with indices from some $I \subseteq [m]$. For $\boldsymbol{f} \in \mathbb{Q}^{1\times m}$, $\boldsymbol{f}A$ is a *linear combination* of the rows $i \in [m]$ of A with $f_i \neq 0$. We call A *linearly independent* if none of its rows is a linear combination of its other rows, and *linearly dependent* otherwise. The *rank of A rank*(A) is the size of a maximal $I \subseteq [m]$ with $A[I]$ linearly independent.

Linear Constraints Let $\boldsymbol{a} = (a_1,\dots,a_n) \in \mathbb{Q}^{1\times n}$, $b \in \mathbb{Q}$ and $\sim \in \{=,\leq,<,\neq\}$ a *relation symbol*. We call $\boldsymbol{ax}$ a *linear term* and $\boldsymbol{ax} \sim b$ a *linear constraint*, which is *weak* if $\sim \in \{=,\leq\}$ and *strict* otherwise. A *system of linear constraints*, or short a *system*, is a non-empty finite set of linear constraints. For most of this paper, we only consider constraints of the form $\boldsymbol{ax} \leq b$. We can write every system $C = \{\boldsymbol{a}_{i,-}\,\boldsymbol{x} \leq b_i \mid i \in [m]\}$ of such constraints in *matrix representation* $A\boldsymbol{x} \leq \boldsymbol{b}$ with suitable $A \in \mathbb{Q}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{Q}^{m\times 1}$. Conversely, every row $\boldsymbol{a}_{i,-}\,\boldsymbol{x} \leq b_i$, $i \in [m]$ of $A\boldsymbol{x} \leq \boldsymbol{b}$ is a linear constraint. Thus, the representations are mostly interchangeable; however, the matrix representation allows redundant rows in contrast to the set notation. As the latter will play a role later on, we will stick to the matrix representation.

Variable Assignment An *assignment* is a function $\alpha : Y \to \mathbb{R}$ with domain $dom(\alpha) = Y \subseteq X$. The *extension* $\alpha[x_i \mapsto r]$ is the assignment with domain $dom(\alpha) \cup \{x_i\}$ such that $\alpha[x_i \mapsto r](x_j) = \alpha(x_j)$ for all $x_j \in dom(\alpha) \setminus \{x_i\}$ and $\alpha[x_i \mapsto r](x_i) = r$. For $Z \subseteq Y$, the *restriction* $\alpha|_Z$ is the assignment with domain Z such that $\alpha|_Z(x_i) = \alpha(x_i)$ for all $x_i \in Z$. We extend these notations to sets of assignments accordingly.

The standard *evaluation* of a linear term t under α is written $\alpha(t)$. We say that α *satisfies* (or is a solution of) a constraint $c = (\boldsymbol{ax} \sim b)$ if $\alpha(a_1x_1 + \dots a_nx_n) \sim b$ holds, and denote this fact by $\alpha \models c$. All solutions of c build its *solution set sol*(c). Similarly, $\alpha \models (A\boldsymbol{x} \leq \boldsymbol{b})$ denotes that α is a common solution of all linear constraints in the system $A\boldsymbol{x} \leq \boldsymbol{b}$. A system is *satisfiable* if it has a common solution, and *unsatisfiable* otherwise. Note that each satisfiable system has also a rational-valued solution.

We will also make use of the following two well-known results.

Theorem 1 (Farkas' Lemma [8]). *Let $A \in \mathbb{Q}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{Q}^{m\times 1}$. Then the system $A\boldsymbol{x} \leq \boldsymbol{b}$ is satisfiable if and only if for all $\boldsymbol{f} \in \mathbb{Q}^{1\times m}$ with $\boldsymbol{f} \geq 0$ and $\boldsymbol{f}A = (0,\dots,0) \in \mathbb{Q}^{1\times n}$ it holds $\boldsymbol{fb} \geq 0$.*

Theorem 2 (Fundamental Theorem of Linear Programming, as in [21]). *Let $A \in \mathbb{Q}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{Q}^{m\times 1}$. Then $A\boldsymbol{x} \leq \boldsymbol{b}$ is satisfiable if and only if there exists a subset $I \subseteq [m]$ such that $A[I]$ is linearly independent, $|I| = rank(A)$, and there exists an assignment $\alpha : X \to \mathbb{R}$ with $\alpha \models (A[I]\boldsymbol{x} = \boldsymbol{b}[I])$ and $\alpha \models (A\boldsymbol{x} \leq \boldsymbol{b})$.*

2.1 Fourier-Motzkin Variable Elimination

The *Fourier-Motzkin variable elimination* (FM) [9, 23] method allows to eliminate any $x_j \in X$ from a system $A\boldsymbol{x} \le \boldsymbol{b}$ by computing $A'\boldsymbol{x} \le \boldsymbol{b'}$ with $\boldsymbol{a'}_{\text{-},j} = 0$ and such that an assignment α is a solution of $A'\boldsymbol{x} \le \boldsymbol{b'}$ if and only if there is $r \in \mathbb{Q}$ so that $\alpha[x_j \mapsto r]$ is a solution of $A\boldsymbol{x} \le \boldsymbol{b}$. Graphically, the solution set of $A'\boldsymbol{x} \le \boldsymbol{b'}$ is the projection of the solutions of $A\boldsymbol{x} \le \boldsymbol{b}$ onto $X \setminus \{x_j\}$.

The idea of the FM method is as follows. For each $i \in [m]$ with $a_{i,j} \ne 0$, the constraint $\boldsymbol{a}_{i,\text{-}}\, \boldsymbol{x} \le b_i$ can be rewritten as either a *lower bound* or an *upper bound* on x_j, denoted in both cases as $bnd_j(\boldsymbol{a}_{i,\text{-}}\, \boldsymbol{x} \le b_i)$:

$$\Big(\sum_{k\in[n]\setminus\{j\}} -\frac{a_{i,k}}{a_{i,j}}\cdot x_k\Big) + \frac{b_i}{a_{i,j}} \le x_j, \;\text{ if } a_{i,j} < 0, \qquad \text{resp.} \qquad x_j \le \Big(\sum_{k\in[n]\setminus\{j\}} -\frac{a_{i,k}}{a_{i,j}}\cdot x_k\Big) + \frac{b_i}{a_{i,j}}, \;\text{ if } a_{i,j} > 0.$$

Definition 1. *For $A \in \mathbb{Q}^{m\times n}$, we define the index sets*

$$I_j^-(A) := \{i \in [m] \mid a_{i,j} < 0\}, \quad I_j^+(A) := \{i \in [m] \mid a_{i,j} > 0\}, \quad \textit{and} \quad I_j^0(A) := \{i \in [m] \mid a_{i,j} = 0\}.$$

$I_j^-(A)$, $I_j^+(A)$ and$I_j^0(A)$ indicate the rows of $A\boldsymbol{x} \le \boldsymbol{b}$ which induce lower bounds, upper bounds and no bounds on x_j, respectively. Due to the density of the reals, there exists a value for x_j that satisfies all bounds if and only if each lower bound is less than or equal to each upper bound. However, since in general the involved bounds are symbolic and thus their values depend on the values of other variables, we cannot directly check this condition. To express this, we let $A'\boldsymbol{x} \le \boldsymbol{b'}$ be defined by the constraint set

$$\{bnd_j(\boldsymbol{a}_{\ell,\text{-}}\, \boldsymbol{x} \le b_\ell) \le bnd_j(\boldsymbol{a}_{u,\text{-}}\, \boldsymbol{x} \le b_u) \mid (\ell,u) \in I_j^-(A) \times I_j^+(A)\} \quad \cup \quad \{\boldsymbol{a}_{i,\text{-}}\, \boldsymbol{x} \le b_i \mid i \in I_j^0(A)\}.$$

In matrix representation, the FM method applies the following transformation:

Definition 2 (Fourier-Motzkin Variable Elimination)**.** *Let $A \in \mathbb{Q}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Q}^{m\times 1}$, and $j \in [n]$. Let further $m' = |I_j^-(A)| \cdot |I_j^+(A)| + |I_j^0(A)|$ and $F \in \mathbb{Q}^{m'\times m}$ be a matrix consisting of exactly the following rows:*[1]

$$-\frac{1}{a_{\ell,j}} \cdot \boldsymbol{e}_{\ell,\text{-}}^{(m)} + \frac{1}{a_{u,j}} \cdot \boldsymbol{e}_{u,\text{-}}^{(m)} \;\textit{ for every pair }\; (\ell,u) \in I_j^-(A) \times I_j^+(A) \qquad \textit{and} \qquad \boldsymbol{e}_{i,\text{-}}^{(m)} \;\textit{ for every }\; i \in I_j^0(A).$$

Then the Fourier-Motzkin variable elimination $\texttt{FM}_j(A\boldsymbol{x} \le \boldsymbol{b})$ *of x_j from the system $A\boldsymbol{x} \le \boldsymbol{b}$ is defined as the system $FA\boldsymbol{x} \le F\boldsymbol{b}$.*

The consistency of $A\boldsymbol{x} \le \boldsymbol{b}$ can be checked by successively eliminating variables $x_n, \dots, x_1$, obtaining intermediate systems $A^{(n-1)}\boldsymbol{x} \le \boldsymbol{b}^{(n-1)}, \dots, A^{(0)}\boldsymbol{x} \le \boldsymbol{b}^{(0)}$. All entries of the transformation matrix F in the definition above are positive, and thus for any $k \in \{0, \dots, n-1\}$ and any row i' in $A^{(k)}\boldsymbol{x} \le \boldsymbol{b}^{(k)}$, there exists $0 \le \boldsymbol{f} \in \mathbb{Q}^{m\times 1}$ s.t. $\boldsymbol{f}A = \boldsymbol{a}_{i',\text{-}}^{(k)}$ and $\boldsymbol{f}\boldsymbol{b} = b_{i'}^{(k)}$, or in short: $\sum_{i\in[m]} f_i \cdot (\boldsymbol{a}_{i,\text{-}}\, \boldsymbol{x} \le b_i) = (\boldsymbol{a}_{i',\text{-}}^{(k)}\boldsymbol{x} \le b_{i'}^{(k)})$. We call this kind of linear combinations *conical combinations*. By Farkas' Lemma (Theorem 1), if $A^{(0)}\boldsymbol{x} \le \boldsymbol{b}^{(0)}$ is unsatisfiable, then so is $A\boldsymbol{x} \le \boldsymbol{b}$. If it is satisfiable, then it is satisfied by the empty assignment, which can be extended successively to a model of $A^{(1)}\boldsymbol{x} \le \boldsymbol{b}^{(1)}, \dots, A^{(n-1)}\boldsymbol{x} \le \boldsymbol{b}^{(n-1)}$ and $A\boldsymbol{x} \le \boldsymbol{b}$.

A major drawback of the Fourier-Motzkin variable elimination is its doubly exponential complexity in time and space w.r.t. the number of eliminated variables. Moreover, many of the generated rows are redundant because they are linear combinations of the other rows, i.e. they could be omitted without changing the solution set of the system. Redundancies might already be contained in the input system, or they arise during the projection operation. While removing all redundancies is expensive, there are efficient methods for removing some redundancies of the latter type, for example Imbert's acceleration theorems [10, 11, 12].

[1]Remember that we use lower case letters for rows of matrices with the respective upper case letter as name. Thus, $\boldsymbol{e}_{i,\text{-}}^{(m)}$ denotes the ith column vector of the identity matrix $E^{(m)}$.

Lemma 1 (Redundancy by Construction)**.** *Let $A \in \mathbb{Q}^{m\times n}, \boldsymbol{b} \in \mathbb{Q}^{m\times 1}$ and $F \in \mathbb{Q}^{m'\times m}$. Let furthermore $A' = FA$, $\boldsymbol{b'} = F\boldsymbol{b}$ and $i \in [m']$. If there exists $\boldsymbol{r} \in \mathbb{Q}^{1\times m'}$ with $\boldsymbol{r} \geq 0$, $r_i = 0$ and $\boldsymbol{r}F = \boldsymbol{f}_{i,-}$ (i.e. the ith row of $A'\boldsymbol{x} \leq \boldsymbol{b'}$ is a conical combination $\boldsymbol{r}FA\boldsymbol{x} \leq \boldsymbol{r}F\boldsymbol{b}$ of the other rows), then that row is redundant in $A'\boldsymbol{x} \leq \boldsymbol{b'}$, i.e. the solution set does not change when omitting it: $sol(A'\boldsymbol{x} \leq \boldsymbol{b'}) = sol(A'[[m'] \setminus \{i\}]\boldsymbol{x} \leq \boldsymbol{b'}[[m'] \setminus \{i\}])$.*

3 FMplex as Variable Elimination Procedure

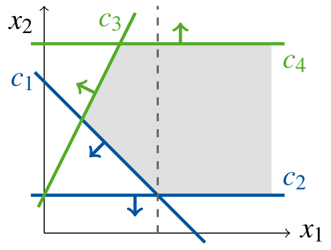
The FM method encodes that none of the lower bounds on some variable x_j in a system $A\boldsymbol{x} \leq \boldsymbol{b}$ is larger than any of its upper bounds. In our *FMplex* method, instead of considering all lower-upper bound combinations at once, we *split the problem into a set of sub-problems* by case distinction either on *which of the lower bounds is the largest* or alternatively on *which of the upper bounds is the smallest*. For splitting on lower bounds, for each lower bound on x_j we consider solutions where this lower bound is maximal under all lower bounds, and at the same time not larger than any of the upper bounds. The upper bound case is analogous. Then $A\boldsymbol{x} \leq \boldsymbol{b}$ is satisfiable if and only if there exists a solution in one of these sub-problems. Asymptotically, these sub-problems are significantly smaller than the systems produced by FM, so that in total our approach produces *at most exponentially* many constraints after iterated application, in contrast to the doubly exponential effort of the FM method.

Formally, if there are no upper or no lower bounds on x_j, then there is no need for case splitting and we follow FM using $\exists x_j.\, A\boldsymbol{x} \leq \boldsymbol{b} \equiv A[I_j^0(A)]\boldsymbol{x} \leq \boldsymbol{b}[I_j^0(A)]$. Otherwise, for the sub-problem when designating $i \in I_j^-(A)$ as largest lower bound, we encode that no other lower bound is larger than the bound induced by row i, and no upper bound is below this bound. Using the set notation for systems, we obtain

$$\{bnd_j(\boldsymbol{a}_{i',-}\,\boldsymbol{x} \leq b_{i'}) \leq bnd_j(\boldsymbol{a}_{i,-}\,\boldsymbol{x} \leq b_i) \mid i' \in I_j^-(A),\, i' \neq i\}$$
$$\cup\{bnd_j(\boldsymbol{a}_{i,-}\,\boldsymbol{x} \leq b_i) \leq bnd_j(\boldsymbol{a}_{i',-}\,\boldsymbol{x} \leq b_{i'}) \mid i' \in I_j^+(A)\} \cup \{\boldsymbol{a}_{i',-}\,\boldsymbol{x} \leq b_{i'} \mid i' \in I_j^0(A)\}.$$

Example 1. *We eliminate x_2 from the system $A\boldsymbol{x} \leq \boldsymbol{b}$ consisting of the lower-bounding constraints c_1 and c_2, and the upper-bounding c_3 and c_4, specified below along with a graphical depiction. The lower bounds $I_2^-(A) = \{1,2\}$ on x_2 are blue, the upper bounds $I_2^+(A) = \{3,4\}$ are green. The solution set is the gray area and the dashed line indicates the split into two sub-problems, namely the cases that c_1 resp. c_2 is a largest lower bound on x_2 and not larger than any upper bound on x_2.*

$$\begin{array}{c} c_1 \\ c_2 \\ c_3 \\ c_4 \end{array} \begin{bmatrix} -1 & -1 \\ 0 & -2 \\ -2 & 1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \leq \begin{bmatrix} -4 \\ -2 \\ 1 \\ 5 \end{bmatrix}$$



The encoding of the c_1-case is given by $(bnd_2(c_2) \leq bnd_2(c_1)) \wedge (bnd_2(c_1) \leq bnd_2(c_3)) \wedge (bnd_2(c_1) \leq bnd_2(c_4))$, which evaluates to $(x_1 \leq 3) \wedge (-3x_1 \leq -3) \wedge (-x_1 \leq 1)$ satisfied by any $x_1 \in [1,3]$, on the left of the dashed line. The case for c_2 evaluates to $(-x_1 \leq -3) \wedge (-2x_1 \leq 0) \wedge (0 \leq 4)$ and is satisfiable on the right of the dashed line. The disjunction of the two formulas then defines exactly those values for x_1 which allow a solution of the initial system.

The construction for the case $i \in I_j^+(A)$ designating i as smallest upper bound is analogous. In matrix representation, these projections are defined by the following transformation:

Definition 3 (Restricted Projection)**.** *Let $A \in \mathbb{Q}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Q}^{m\times 1}$ and $j \in [n]$.*

- *If $I_j^-(A) \neq \emptyset$ and $I_j^+(A) \neq \emptyset$, then for any $i \in I_j^-(A) \cup I_j^+(A)$ we fix $F \in \mathbb{Q}^{(m-1)\times m}$ arbitrarily but deterministically to consist of exactly the following rows:*

$$\frac{1}{a_{i,j}} \cdot \boldsymbol{e}^{(m)}_{i,\text{-}} - \frac{1}{a_{i',j}} \cdot \boldsymbol{e}^{(m)}_{i',\text{-}} \text{ for every } i' \in I_j^-(A) \setminus \{i\},$$

$$-\frac{1}{a_{i,j}} \cdot \boldsymbol{e}^{(m)}_{i,\text{-}} + \frac{1}{a_{i',j}} \cdot \boldsymbol{e}^{(m)}_{i',\text{-}} \text{ for every } i' \in I_j^+(A) \setminus \{i\}, \qquad \text{and} \qquad \boldsymbol{e}^{(m)}_{i',\text{-}} \text{ for every } i' \in I_j^0(A).$$

Then the restricted projection $P_{j,i}(A\boldsymbol{x} \le \boldsymbol{b})$ *of x_j w.r.t. the row i from the system $A\boldsymbol{x} \le \boldsymbol{b}$ is defined as the system $FA\boldsymbol{x} \le F\boldsymbol{b}$. We call F the* projection matrix *corresponding to $P_{j,i}(A\boldsymbol{x} \le \boldsymbol{b})$.*

- *If $I_j^-(A) = \emptyset$ or $I_j^+(A) = \emptyset$, then we define the projection matrix $F \in \mathbb{Q}^{|I_j^0(A)|\times m}$ to have exactly one row $\boldsymbol{e}^{(m)}_{i',\text{-}}$ for each $i' \in I_j^0(A)$, and define $P_{j,\perp}(A\boldsymbol{x} \le \boldsymbol{b})$ as $FA\boldsymbol{x} \le F\boldsymbol{b}$.*

The following lemma states a crucial result for our method: The solutions of the restricted projections for all lower (or all upper) bounds of a variable exactly cover the projection of the entire solution set.

Lemma 2. *Let $A \in \mathbb{Q}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Q}^{m\times 1}$, $j \in [n]$ and $I \in \{I_j^-(A), I_j^+(A)\}$. If $I_j^-(A) \neq \emptyset$ and $I_j^+(A) \neq \emptyset$, then*

$$sol(A\boldsymbol{x} \le \boldsymbol{b})|_{X\setminus\{x_j\}} = \bigcup_{i\in I} sol(P_{j,i}(A\boldsymbol{x} \le \boldsymbol{b})).$$

Otherwise ($I_j^-(A) = \emptyset$ or $I_j^+(A) = \emptyset$), it holds $sol(A\boldsymbol{x} \le \boldsymbol{b})|_{X\setminus\{x_j\}} = sol(P_{j,\perp}(A\boldsymbol{x} \le \boldsymbol{b}))$.

Proof. The case $I_j^-(A) = \emptyset$ or $I_j^+(A) = \emptyset$ follows from the correctness of FM. Assume $I = I_j^-(A)$, the case $I = I_j^+(A)$ is analogous.

$\supseteq$: Let $i \in I_j^-(A)$ and $\alpha \models P_{j,i}(A\boldsymbol{x} \le \boldsymbol{b})$, then for all $\ell \in I_j^-(A)$, $u \in I_j^+(A)$ it holds $\alpha(bnd_j(\boldsymbol{a}_{\ell,\text{-}}\,\boldsymbol{x} \le b_\ell)) \le \alpha(bnd_j(\boldsymbol{a}_{i,\text{-}}\,\boldsymbol{x} \le b_i)) \le \alpha(bnd_j(\boldsymbol{a}_{u,\text{-}}\,\boldsymbol{x} \le b_u))$. Thus, $\alpha[x_j \mapsto \alpha(bnd_j(\boldsymbol{a}_{i,\text{-}}\,\boldsymbol{x} \le b_i))] \models A\boldsymbol{x} \le \boldsymbol{b}$.

$\subseteq$: Let $\alpha \models A\boldsymbol{x} \le \boldsymbol{b}$ and $i = \arg\max_{\ell \in I_j^-(A)}(\alpha(bnd_j(\boldsymbol{a}_{\ell,\text{-}}\,\boldsymbol{x} \le b_\ell)))$, then for all $u \in I_j^+(A)$ it holds $\alpha(bnd_j(\boldsymbol{a}_{i,\text{-}}\,\boldsymbol{x} \le b_i)) \le \alpha(bnd_j(\boldsymbol{a}_{u,\text{-}}\,\boldsymbol{x} \le b_u))$ and thus $\alpha \models P_{j,i}(A\boldsymbol{x} \le \boldsymbol{b})$. □

Definition 4 (FMplex Variable Elimination)**.** *For $A \in \mathbb{Q}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Q}^{m\times 1}$, $j \in [n]$ and $* \in \{-,+\}$, we define*

$$\texttt{FMP}^*_j(A\boldsymbol{x} \le \boldsymbol{b}) = \begin{cases} \{P_{j,i}(A\boldsymbol{x} \le \boldsymbol{b}) \mid i \in I_j^*(A)\} & \text{if } I_j^-(A) \neq \emptyset \text{ and } I_j^+(A) \neq \emptyset \\ \{P_{j,\perp}(A\boldsymbol{x} \le \boldsymbol{b})\} & \text{otherwise.} \end{cases}$$

The FMplex elimination defines a set of restricted projections which can be composed to the full projection according to Lemma 2. Lifting this from sets to logic naturally results in the following theorem which demonstrates the usage of our method.

Theorem 3. *Let $A \in \mathbb{Q}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Q}^{m\times 1}$, and $j \in [n]$. Then*

$$\exists x_j.\, A\boldsymbol{x} \le \boldsymbol{b} \quad \equiv \quad \bigvee\nolimits_{S \in \texttt{FMP}^+_j(A\boldsymbol{x} \le \boldsymbol{b})} S \quad \equiv \quad \bigvee\nolimits_{S \in \texttt{FMP}^-_j(A\boldsymbol{x} \le \boldsymbol{b})} S.$$

For eliminating multiple variables, we iteratively apply $\texttt{FMP}^-$ or $\texttt{FMP}^+$ to each restricted projection resulting from the previous elimination step. Note that we can choose the next variable to be eliminated as well as the variant independently in every branch.

Example 2. *We continue Example 1, from which we eliminated x_2 and now want to eliminate x_1:*

$$\begin{aligned}\exists x_1.\,\exists x_2.\,A\boldsymbol{x} \le \boldsymbol{b} &\equiv \exists x_1.\bigvee\nolimits_{S\in \texttt{FMP}_2^-(A\boldsymbol{x}\le\boldsymbol{b})} S\\ &\equiv \exists x_1.\,(x_1 \le 3 \wedge -3x_1 \le -3 \wedge -x_1 \le 1)\vee \exists x_1.\,(-x_1 \le -3 \wedge -2x_1 \le 0 \wedge 0 \le 4)\end{aligned}$$

We eliminate the two quantifiers for x_1 separately, using

$$\begin{aligned}\texttt{FMP}_1^-(x_1 \le 3 \wedge -3x_1 \le -3 \wedge -x_1 \le 1) &= \{(0\le 2 \wedge 0 \le 2),(0 \le -2 \wedge 0 \le 4)\} \text{ and}\\ \texttt{FMP}_1^-(-x_1 \le -3 \wedge -2x_1 \le 0 \wedge 0 \le 4) &= \{(0 \le 4)\}\end{aligned}$$

giving us the final result $\exists x_1.\,\exists x_2.\,A\boldsymbol{x}\le\boldsymbol{b} \equiv ((0\le 2\wedge 0\le 2)\vee(0\le 4\wedge 0\le -2))\vee(0\le 4)$.

We analyze the complexity in terms of the number of new rows (or constraints) that are constructed during the elimination of all variables:

Theorem 4 (Complexity of FMP)**.** *Let $A \in \mathbb{Q}^{m\times n}$, and $\boldsymbol{b}\in\mathbb{Q}^{m\times 1}$. When eliminating n variables from $A\boldsymbol{x}\le\boldsymbol{b}$, the $\texttt{FMP}^-$ method constructs $\mathcal{O}(n\cdot m^{n+1})$ new rows.*

Proof. The number $N(m,n)$ of constructed rows is maximal if the system consists only of lower bounds and one upper bound. Then, $\texttt{FMP}^-$ yields $m-1$ new systems of size $m-1$, from which $n-1$ variables need to be eliminated; thus $N(m,n)\le(m-1)\cdot((m-1)+N(m-1,n-1))$. With $k=\min(n,m)$, we obtain $N(m,n)\le \sum_{i=1}^{k}(m-i)\cdot\prod_{j=1}^{i}(m-j)\le n\cdot m^{n+1}$. □

While still exponential, this bound is considerably better than the theoretical doubly exponential worst-case complexity of the FM method. Shortly speaking, FMplex trades one exponential step at the cost of the result being a decomposition into multiple partial projections. However, there are systems for which FMplex produces strictly more rows than the FM method: In the worst case from the above proof, FM obtains a single system of the same size as each of the sub-problems computed by $\texttt{FMP}^-$. Although in this case, we could simply employ $\texttt{FMP}^+$ instead, it is unclear whether there exists a rule for employing $\texttt{FMP}^-$ or $\texttt{FMP}^+$ that never produces more constraints than FM.

Like FM, FMplex keeps redundancies from the input throughout the algorithm, thus there might be identical rows in the same or across different sub-problems. But in contrast to FM, FMplex does not introduce any redundancies by construction in the sense of Lemma 1.

Theorem 5. *Let $A\in\mathbb{Q}^{m\times n}$, $\boldsymbol{b}\in\mathbb{Q}^{m\times 1}$ and $k\in[m]$. Assume $(A^{(0)}\boldsymbol{x}\le\boldsymbol{b}^{(0)})=(A\boldsymbol{x}\le\boldsymbol{b})$ and for all $j\in[k]$, let $(A^{(j)}\boldsymbol{x}\le\boldsymbol{b}^{(j)})\in\texttt{FMP}_j^-(A^{(j-1)}\boldsymbol{x}\le\boldsymbol{b}^{(j-1)})\cup\texttt{FMP}_j^+(A^{(j-1)}\boldsymbol{x}\le\boldsymbol{b}^{(j-1)})$. Let $F^{(1)},\dots,F^{(k)}$ be the respective projection matrices, and $F=F^{(k)}\cdot\ldots\cdot F^{(1)}$. Then F is linearly independent.*

Proof. By definition, the projection matrices are linearly independent, and thus so is their product F. □

4 FMplex as Satisfiability Checking Procedure

A formula is satisfiable if and only if eliminating all variables (using any quantifier elimination method such as FM or FMplex) yields a tautology. However, FMplex computes smaller sub-problems whose satisfiability implies the satisfiability of the original problem. Therefore, we do not compute the whole projection at once, but explore the decomposition using a depth-first search. The resulting search tree has the original system as root, and each node has as children the systems resulting from restricted projections. The original system is satisfiable if and only if a leaf without any trivially false constraints exists.

$$\begin{tikzcd}
 & & A\boldsymbol{x} \leq \boldsymbol{b} \arrow[dl, no head] \arrow[dr, no head] & \\
 & P_{2,1}(A\boldsymbol{x} \leq \boldsymbol{b}) \arrow[dl, no head] \arrow[dr, no head] & & P_{2,2}(A\boldsymbol{x} \leq \boldsymbol{b}) \arrow[d, no head] \\
(0 \leq 2) \wedge (0 \leq 2) & & (0 \leq 4) \wedge (0 \leq -2) & (0 \leq 4)
\end{tikzcd}$$

Figure 2: The search tree corresponding to Example 2. The very first leaf (bottom left) is already satisfiable, meaning that the rest would not need to be computed.

An example is depicted in Figure 2. We start with a basic version of the algorithm and then examine how the search tree can be pruned, resulting in two variants; all versions are given in Algorithm 1.

An important observation is that we can decide independently for each node of the search tree, which variable to eliminate next and whether to branch on lower or on upper bounds.

Definition 5 (Branch Choices)**.** *The set of* branch choices *for a system $A\boldsymbol{x} \leq \boldsymbol{b}$ is*

$$\begin{aligned}
branch\_choices(A\boldsymbol{x} \leq \boldsymbol{b}) = &\{\{(x_j, i) \mid i \in I_j^-(A)\} \mid j \in [n] \wedge I_j^-(A) \neq \emptyset \wedge I_j^+(A) \neq \emptyset\} \\
\cup &\{\{(x_j, i) \mid i \in I_j^+(A)\} \mid j \in [n] \wedge I_j^-(A) \neq \emptyset \wedge I_j^+(A) \neq \emptyset\} \\
\cup &\{\{(x_j, \bot)\} \mid j \in [n] \wedge (I_j^-(A) = \emptyset \vee I_j^+(A) = \emptyset)\}.
\end{aligned}$$

For an initial input $\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}$ with $\widehat{m}$ rows, we define the depth-first search using the recursive method $\texttt{FMplex}(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}; A\boldsymbol{x} \leq \boldsymbol{b}, F)$ in Algorithm 1a where $A\boldsymbol{x} \leq \boldsymbol{b}$ is the currently processed sub-problem in the recursion tree. We track the relation of $A\boldsymbol{x} \leq \boldsymbol{b}$ to $\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}$ in terms of linear combinations using the parameter F. The initial call is defined as $\texttt{FMplex}(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}) = \texttt{FMplex}(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}; \widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}, E^{(\widehat{m})})$. We allow that $A\boldsymbol{x} \leq \boldsymbol{b}$ contains identical rows when they are obtained in different ways (which is reflected by F). We need to keep these duplicates for proving the results of this section.

Solutions If a trivially satisfiable node is found, the algorithm constructs an assignment starting with the empty assignment and extends it in reverse order in which the variables were eliminated. For every variable x_j, a value is picked above all lower and below all upper bounds on x_j evaluated at the underlying assignment. By the semantics of the projection, the value of the designated (largest lower or smallest upper) bound on x_j is suitable.

Conflicts We distinguish inconsistencies in $A\boldsymbol{x} \leq \boldsymbol{b}$ by the following notions: We call a row i of $A\boldsymbol{x} \leq \boldsymbol{b}$ a *conflict* if it is of the form $\boldsymbol{a}_{i,\text{-}} = \boldsymbol{0}^{(n)}$ with $b_i < 0$. We call the conflict *global* if $\boldsymbol{f}_{i,\text{-}} \geq 0$ and *local* otherwise. In case of a global conflict, Farkas' Lemma allows to deduce the unsatisfiability of $\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}$, thus stopping the search before the whole search tree is generated. Then a set of conflicting rows K of the input system corresponding to $\boldsymbol{f}_{i,\text{-}}$ is returned. In particular, the set $\{\widehat{\boldsymbol{a}}_{j,\text{-}}\, \boldsymbol{x} \leq \widehat{b}_j \mid f_{i,j} \neq 0\}$ is a minimal unsatisfiable subset of the constraints in $\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}$. In case of a local conflict, we simply continue to explore the search tree. The algorithm returns *PARTIAL-UNSAT* to indicate that $A\boldsymbol{x} \leq \boldsymbol{b}$ is unsatisfiable, but the unsatisfiability of $\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}$ cannot be derived. This approach, formalized in Algorithm 1a, guarantees that the initial call will never return *PARTIAL-UNSAT*; we always find either a global conflict or a solution.

The correctness and completeness of `FMplex` follows from Theorem 3 and Theorem 6.

Theorem 6. *Let $\widehat{A} \in \mathbb{Q}^{\widehat{m} \times n}$, and $\widehat{\boldsymbol{b}} \in \mathbb{Q}^{\widehat{m}} \times 1$. Then $\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}$ is unsatisfiable if and only if the call $\mathtt{FMplex}(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}})$ to Algorithm 1a terminates with a global conflict.*

Algorithm 1: $\texttt{FMplex}(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}; A\boldsymbol{x} \leq \boldsymbol{b}, F, \boxed{\mathcal{N}, I}, \texttt{lvl}, \texttt{bt\_lvl})$

Algorithm 1a The base method consists of the plain (unframed and unfilled) parts.

Algorithm 1b Consists of the base method and the $\boxed{\text{framed parts}}$.

Algorithm 1c Consists of the base method, the $\boxed{\text{framed parts}}$ and the filled boxes.

```
Data   : $\widehat{A} \in \mathbb{Q}^{\widehat{m}\times n}$, $\widehat{\boldsymbol{b}} \in \mathbb{Q}^{\widehat{m}}$
Input  : $A \in \mathbb{Q}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Q}^{m}$, $F \in \mathbb{Q}^{m\times \widehat{m}}$ s.t. $F\widehat{A} = A$ and $F\widehat{\boldsymbol{b}} = \boldsymbol{b}$, $\boxed{\mathcal{N} \subseteq [\widehat{m}], I \subseteq [\widehat{m}]}$,
         $\texttt{lvl} \in [n] \cup \{0\}$, and $\texttt{bt\_lvl} : [m] \to [n] \cup \{0\}$
Output: $(SAT, \alpha)$ with $\alpha \models A\boldsymbol{x} \leq \boldsymbol{b}$, or $(UNSAT, K)$ where $K \subseteq [\widehat{m}]$, or
         $(PARTIAL\text{-}UNSAT, l, K)$ where $l \in [n]$ and $K \subseteq [\widehat{m}]$

1  if $A = 0 \land \boldsymbol{b} \geq 0$ then return $(SAT, ())$
2  if $\exists i \in [m].\ \boldsymbol{a}_{i,-} = 0 \land b_i < 0 \land \boldsymbol{f}_{i,-} \geq 0$ then return $(UNSAT, \{i' \mid f_{i,i'} \neq 0\})$
3  if $\exists i \in [m].\ \boldsymbol{a}_{i,-} = 0 \land b_i < 0 \land \boldsymbol{f}_{i,-} \not\geq 0$ then
4      $i := \arg\min_{i\in[m]}\{\texttt{bt\_lvl}(i) \mid \boldsymbol{a}_{i,-} = 0 \land b_i < 0\}$
5      return $(PARTIAL\text{-}UNSAT, \texttt{bt\_lvl}(i) - 1, \{i' \mid f_{i,i'} \neq 0\})$
6  $K = \emptyset$
7  choose $V \in branch\_choices(A\boldsymbol{x} \leq \boldsymbol{b}, \boxed{\{\mathcal{B}^{-1}_{\mathcal{N},F}(i) \mid i \in I\}})$
8  foreach $(x_j, i) \in V$ do
9      compute $A'\boldsymbol{x} \leq \boldsymbol{b}' := P_{j,i}(A\boldsymbol{x} \leq \boldsymbol{b})$ with projection matrix $F'$ and backtrack levels $\texttt{bt\_lvl}'$
10     $\boxed{\mathcal{N}' := \mathcal{N} \cup \{\mathcal{B}_{\mathcal{N},F}(i)\} \text{ if } i \neq \bot \text{ else } \mathcal{N}}$
11     switch $\texttt{FMplex}\,(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}; A'\boldsymbol{x} \leq \boldsymbol{b}', F'F, \boxed{\mathcal{N}', I}, \texttt{lvl}+1, \texttt{bt\_lvl}')$ do
12         case $(UNSAT, K')$ do return $(UNSAT, K')$
13         case $(SAT, \alpha)$ do return $(SAT, \alpha[x_j \mapsto r])$ for a suitable $r \in \mathbb{Q}$
14         case $(PARTIAL\text{-}UNSAT, l, K')$ do
15             if $l < \texttt{lvl}$ then return $(PARTIAL\text{-}UNSAT, l, K')$
16             else $K = K \cup K'$
17     $\boxed{I := I \cup \{\mathcal{B}_{\mathcal{N},F}(i)\}}$
18 if $\texttt{lvl} = 0$ then return $(UNSAT, K)$
19 return $(PARTIAL\text{-}UNSAT, \texttt{lvl-1}, K)$
```

Proof Idea for Theorem 6. If $\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}$ is unsatisfiable, then there exists a minimal unsatisfiable subset $\widehat{K}$ of the corresponding constraints. We construct a path in the search tree induced by Algorithm 1a yielding a conflict that is a linear combination of $\widehat{K}$. As $\widehat{K}$ is minimal, the linear combination is positive, i.e. the conflict is global. The other direction of the equivalence follows immediately with Farkas' Lemma. Consult the extended version for a detailed proof. □

4.1 Avoiding Redundant Checks

We observe that each row i in a sub-problem $A\boldsymbol{x} \leq \boldsymbol{b}$ in the recursion tree of `FMplex`$(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}})$ corresponds to a row $\hat{\imath}$ in $\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}$ in the sense that it is a linear combination of the rows $\{\hat{\imath}\} \cup \mathcal{N}$ of $\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}$, where $\mathcal{N} \subseteq [\widehat{m}]$ corresponds to the lower/upper bounds designated as largest/smallest one to compute $A\boldsymbol{x} \leq \boldsymbol{b}$:

Theorem 7. *Let $\widehat{A} \in \mathbb{Q}^{\widehat{m}\times n}$ and $\widehat{\boldsymbol{b}} \in \mathbb{Q}^{\widehat{m}\times 1}$. Let FMplex$(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}; A\boldsymbol{x} \leq b, F)$ be a call in the recursion tree of the call FMplex$(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}})$ to Algorithm 1a , where $A \in \mathbb{Q}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{Q}^{m\times 1}$ (by construction $m \leq \widehat{m}$).*
Then there exists a set $\mathcal{N} \subseteq [\widehat{m}]$ such that

1. *$A\boldsymbol{x} \leq \boldsymbol{b}$ is satisfiable if and only if $(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}) \wedge (\widehat{A}[\mathcal{N}]\boldsymbol{x} = \widehat{\boldsymbol{b}}[\mathcal{N}])$ is satisfiable,*
2. *there exists an injective mapping $\mathcal{B}_{\mathcal{N},F} : [m] \to [\widehat{m}], i \mapsto \hat{\imath}$ with $\{\hat{\imath}\} = \{i' \in [\widehat{m}] \mid f_{i,i'} \neq 0\} \setminus \mathcal{N}$.*

Proof Idea. The statement follows with a straight forward induction over the elimination steps, where the original row corresponding to the chosen bound is added to $\mathcal{N}$, and $\mathcal{B}_{\mathcal{N},F}$ keeps track of which constraint corresponds to which original row. Consult the extended version for a detailed proof. □

We call the above defined set $\mathcal{N}$ the *non-basis*, inspired from the analogies to the simplex algorithm (discussed in Section 5.1). By the above theorem, the order in which a non-basis is constructed has no influence on the satisfiability of the induced sub-problem. In particular:

Theorem 8. *Let $A \in \mathbb{Q}^{m\times n}$, $\boldsymbol{b} \in \mathbb{Q}^{m\times 1}$, $j \in [n]$, and let $i, i' \in [m]$ be row indices with $a_{i,j} \neq 0$ and $a_{i',j} \neq 0$. If $P_{j,i}(A\boldsymbol{x} \leq \boldsymbol{b})$ is unsatisfiable, then $P_{j,i'}(A\boldsymbol{x} \leq \boldsymbol{b}) \wedge (\boldsymbol{a}_{i,-}\,\boldsymbol{x} = b_i)$ is unsatisfiable.*

Proof. By Theorem 7, if $P_{j,i}(A\boldsymbol{x} \leq \boldsymbol{b})$ is unsatisfiable, then $(A\boldsymbol{x} \leq \boldsymbol{b}) \wedge (\boldsymbol{a}_{i,-}\,\boldsymbol{x} = \boldsymbol{b_i})$ is unsatisfiable, and trivially $(A\,\boldsymbol{x} \leq \boldsymbol{b}) \wedge (\boldsymbol{a}_{i,-}\,\boldsymbol{x} = \boldsymbol{b_i}) \wedge (\boldsymbol{a}_{i',-}\,\boldsymbol{x} = \boldsymbol{b}_{i'})$ is unsatisfiable as well. Using Theorem 7 in the other direction yields that $P_{j,i'}(A\boldsymbol{x} \leq \boldsymbol{b}) \wedge (\boldsymbol{a}_{i,-}\,\boldsymbol{x} = \boldsymbol{b_i})$ is unsatisfiable. □

This suggests that if `FMplex`$(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}; A\boldsymbol{x} \leq \boldsymbol{b}, F)$ with non-basis $\mathcal{N}$ has a child call for row i which does not return *SAT*, then no other call in the recursion tree of `FMplex`$(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}; A\boldsymbol{x} \leq \boldsymbol{b}, F)$ where the corresponding non-basis contains $\mathcal{B}_{\mathcal{N},F}(i)$ will return *SAT* either. Hence, we can ignore $\mathcal{B}_{\mathcal{N},F}(i)$ as designated bound in the remaining recursion tree of `FMplex`$(\widehat{A}\boldsymbol{x} \leq \widehat{\boldsymbol{b}}; A\boldsymbol{x} \leq \boldsymbol{b}, F)$.

Example 3. *Consider the system from Example 1, with an additional constraint $c_5 : (-x_2 \leq 0)$. If c_5 is tried first as greatest lower bound on x_2, then the combination with $c_2 : (-2x_2 \leq -2)$ yields the local conflict $\frac{1}{2}c_2 - c_5 = (0 \leq -1)$. Thus, this branch and, due to Theorem 8, any non-base containing row 5 yields an unsatisfiable system.*

Next, we try c_1 as greatest lower bound on x_2 resulting in the combinations $\frac{1}{2}c_2 - c_1 = (x_1 \leq 3)$, $c_5 - c_1 = (x_1 \leq 4)$, $c_1 + c_3 = (-3x_1 \leq -3)$ and $c_1 + c_4 = (-x_1 \leq 1)$ and corresponding non-base $\{1\}$.

If we now choose $(x_1 \leq 4)$ as smallest upper bound on x_1, leading to the non-base $\{1,5\}$, another local conflict occurs: $(x_1 \leq 3) - (x_1 \leq 4) = (0 \leq -1)$. As 5 is contained in the non-base, we could know beforehand that this would happen and thus avoid computing this branch.

We update the `FMplex` algorithm as shown in Algorithm 1b using the following definition:

Definition 6. *The set of* branch choices *for $A\boldsymbol{x} \le \boldsymbol{b}$ with m rows w.r.t. $I \subseteq [m]$ is*

$$\begin{aligned} branch\_choices(A\boldsymbol{x} \le \boldsymbol{b}, I) = \quad & \{\{(x_j, i) \mid i \in I_j^-(A) \setminus I\} \mid j \in [n] \wedge I_j^-(A) \neq \emptyset \wedge I_j^+(A) \neq \emptyset\} \\ & \cup \{\{(x_j, i) \mid i \in I_j^+(A) \setminus I\} \mid j \in [n] \wedge I_j^-(A) \neq \emptyset \wedge I_j^+(A) \neq \emptyset\} \\ & \cup \{\{(x_j, \bot)\} \mid j \in [n] \wedge (I_j^-(A) = \emptyset \vee I_j^+(A) = \emptyset)\}. \end{aligned}$$

It is easy to see that this modification prevents visiting non-basis twice in the following sense:

Theorem 9. *Let* $\texttt{FMplex}(\widehat{A}\boldsymbol{x} \le \widehat{\boldsymbol{b}}; A\boldsymbol{x} \le \boldsymbol{b}, \_, \mathcal{N}, \_)$ *and* $\texttt{FMplex}(\widehat{A}\boldsymbol{x} \le \widehat{\boldsymbol{b}}; A'\boldsymbol{x} \le \boldsymbol{b'}, \_, \mathcal{N}', \_)$ *be two calls in the recursion tree of a call to Algorithm 1b. Then either $\mathcal{N} \neq \mathcal{N}'$ or one of the systems occurs in the subtree below the other and only unbounded variables are eliminated between them (i.e. one results from the other by deleting some rows).* □

Theorem 10 states that, still, Algorithm 1b always terminates with *SAT* or a global conflict. This follows by a slight modification of the proof of Theorem 6, presented in the extended version of this paper.

Theorem 10. *Let $\widehat{A} \in \mathbb{Q}^{\widehat{m} \times n}$, and $\widehat{\boldsymbol{b}} \in \mathbb{Q}^{\widehat{m} \times 1}$. Then $\widehat{A}\boldsymbol{x} \le \widehat{\boldsymbol{b}}$ is unsatisfiable if and only if the call* $\texttt{FMplex}(\widehat{A}\boldsymbol{x} \le \widehat{\boldsymbol{b}})$ *to Algorithm 1b terminates with a global conflict.* □

4.2 Backtracking of Local Conflicts

So far, we ignored local conflicts that witness the unsatisfiability of a given sub-problem. In this section, we will cut off parts of the search tree based on local conflicts and examine the theoretical implications.

We applied Farkas' Lemma on conflicting rows in some sub-problem that are positive linear combinations of rows from the input system. We can also apply Farkas' Lemma to conflicting rows which are positive linear combinations of some *intermediate* system to conclude the unsatisfiability of the latter. Whenever such a conflict occurs, we can backtrack to the parent system of that unsatisfiable system. Instead of tracking the linear combinations of every row in terms of the rows of each preceding intermediate system, we can do an incomplete check: If a conflicting row was computed only by addition operations, then it is a positive linear combination of the involved rows. Thus, we assign to every intermediate system a level, representing its depth in the search tree and store for every row the level where the last subtraction was applied to the row (i.e. a lower (upper) bound was subtracted from another lower (upper) bound). If a row is conflicting, we can conclude that the intermediate system at this level is unsatisfiable, thus we can jump back to its parent.

Assume the current system is $A\boldsymbol{x} \le \boldsymbol{b}$ at level `lvl` with m rows whose backtracking levels are stored in $\texttt{bt\_lvl} : [m] \to ([n] \cup \{0\})$. If $\texttt{lvl} = 0$, then `bt_lvl` maps all values to 0. When computing $P_{j,i}(A\boldsymbol{x} \le \boldsymbol{b})$ for some x_j and i with projection matrix F, the backtracking levels of the rows in the resulting system $FA\boldsymbol{x} \le F\boldsymbol{b}$ are stored in `bt_lvl'` where for each row i''

$$\texttt{bt\_lvl}'(i'') := \begin{cases} \max\{\texttt{bt\_lvl}(i), \texttt{bt\_lvl}(i')\} & \text{if } f_{i'',i}, f_{i'',i'} > 0 \text{ and } f_{i'',k} = 0,\ k \notin \{i, i'\} \\ \texttt{lvl} & \text{otherwise.} \end{cases}$$

The backtracking scheme is given in Algorithm 1c , which returns additional information in the *PARTIAL-UNSAT* case, that is the backtrack level l of the given conflict, and a (possibly non-minimal) unsatisfiable subset K.

Theorem 11. *Let* $\texttt{FMplex}(\_; A\boldsymbol{x} \le \boldsymbol{b}, \_, \_, \_, \texttt{lvl}, \_)$ *be a call to Algorithm 1c , and consider a second call* $\texttt{FMplex}(\_; A'\boldsymbol{x} \le \boldsymbol{b'}, \_, \_, \_, \_, \texttt{bt\_lvl}')$ *in the recursion tree of the first call. If $A'\boldsymbol{x} \le \boldsymbol{b'}$ has a local conflict in a row i with $\texttt{bt\_lvl}'(i) = \texttt{lvl}$, then $A\boldsymbol{x} \le \boldsymbol{b}$ is unsatisfiable.*

Proof. By construction of `bt_lvl'`, $\boldsymbol{a}'_{i,-}\,\boldsymbol{x} \le b'_i$ is a positive sum of rows from $A\boldsymbol{x} \le \boldsymbol{b}$, i.e. there exists an $\boldsymbol{f} \in \mathbb{Q}^{1\times m}$ such that $(\boldsymbol{f}A\boldsymbol{x} \le \boldsymbol{f}\boldsymbol{b}) = (\boldsymbol{a}'_{i,-}\,\boldsymbol{x} \le b'_i)$. Then by Farkas' Lemma, $A\boldsymbol{x} \le \boldsymbol{b}$ is unsatisfiable. □

While it is complete and correct, Algorithm 1c does not always terminate with a *global* conflict (i.e. Theorem 6 does not hold any more), even if we do not ignore any rows (i.e. omit Line 17):

$$\begin{bmatrix} 0 & 0 & -1 \\ 1 & -1 & -1 \\ 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \le \begin{bmatrix} 0 \\ 0 \\ -1 \\ -1 \\ 0 \end{bmatrix}$$

Example 4. *We use Algorithm 1c to eliminate variables with the static order x_3,x_2,x_1 from the system on the right, always branching on lower bounds. We first choose row* 1 *as greatest lower bound on x_3. Rows* 3 *and* 4 *are retained as they do not contain x_3 and the combination of row* 1 *with row* 5 *is positive, so these constraints have backtrack level* 0*.*
The combination with row 2 *has backtrack level* 1 *because both rows are lower bounds. Using this constraint as greatest lower bound on x_2 and combining it with row* 4 *leads to a local conflict with backtrack level* 1*. This means that the call at level* 1 *is unsatisfiable and thus we backjump to level* 0*.*

The second branch is visited, leading to the non-basis $\mathcal{N} = \{2,5,1\}$ after three steps, where a local *conflict lets us backjump to level* 0 *again. As there are no more lower bounds on x_3, the algorithm returns UNSAT without finding a global conflict.*

5 Relation to Other Methods

5.1 Simplex Algorithm

The simplex method [6, 18] is an algorithm for linear optimization over the reals and is able to solve *linear programs*. The *general simplex* [7] is an adaption for checking the satisfiability of systems of linear constraints. We illustrate its idea for the weak case.

Remind that given a system $A\boldsymbol{x} \le \boldsymbol{b}$ with m rows, by the fundamental theorem of linear programming (Theorem 2), $A\boldsymbol{x} \le \boldsymbol{b}$ is satisfiable if and only if there exists some maximal subset $\mathcal{N} \subseteq [m]$ such that $A[\mathcal{N}]$ is linearly independent and $A\boldsymbol{x} \le \boldsymbol{b} \cup A[\mathcal{N}]\boldsymbol{x} = \boldsymbol{b}[\mathcal{N}]$ is satisfiable - the latter can be checked algorithmically using Gaussian elimination, resulting in a system where each variable is replaced by bounds induced by the rows $\mathcal{N}$. This system along with the information which element in $\mathcal{N}$ was used to eliminate which variable is called a *tableau*. The idea of the simplex method is to do a local search on the set $\mathcal{N}$ (called *non-basis*), that is, we replace some $i \in \mathcal{N}$ (*leaving variable*) by some $i' \in [m] \setminus \mathcal{N}$ (*entering variable*) obtaining $\mathcal{N}' := \mathcal{N} \cup \{i'\} \setminus \{i\}$ such that $A[\mathcal{N}']$ is still linearly independent. The clou is that the tableau representing $(A\boldsymbol{x} \le \boldsymbol{b}) \wedge (A[\mathcal{N}]\boldsymbol{x} = \boldsymbol{b}[\mathcal{N}])$ can be efficiently transformed into $(A\boldsymbol{x} \le \boldsymbol{b}) \wedge (A[\mathcal{N}']\boldsymbol{x} = \boldsymbol{b}[\mathcal{N}'])$ (called *pivot operation*), and progress of the local search can be achieved by the choice of i and i'. These local search steps are performed until a satisfying solution is found, or a conflict is found. These conflicts are detected using Farkas' Lemma (Theorem 1), i.e. a row in the tableau induces a trivially false constraint and is a positive linear combination of some input rows.

As suggested by Theorem 7, there is a strong correspondence between a tableau of the simplex algorithm and the intermediate systems constructed in FMplex. More precisely, if a non-basis of a simplex tableau is equal to the non-basis of a leaf system of Algorithm 1a, then the tableau is satisfiable if and only if the FMplex system is satisfiable. In fact, we could use the same data structure to represent the algorithmic states. Comparing the two algorithms structurally, FMplex explores the search space in a tree-like structure using backtracking, while simplex can jump between neighbouring leaves directly.

The idea for Algorithm 1b that excludes visiting the same non-basis in fact arose from the analogies between the two methods. Further, we observe a potential advantage of FMplex: Simplex has more non-bases reachable from a given initial state than the leaves of the search tree of FMplex, as FMplex needs only to explore all lower or all upper bounds of a variable while simplex does local improvements blindly. Heuristically, simplex cuts off large parts of its search space and we expect it often visits fewer non-bases than FMplex - however, as the pruning done by FMplex is by construction of the algorithm, we believe that there might be combinatorially hard instances on which it is more efficient than simplex.

5.2 Virtual Substitution Method

Virtual substitution [20, 27] admits quantifier elimination for real arithmetic formulas. Here, we consider its application on existentially quantified conjunctions of linear constraints.

The underlying observation is that the satisfaction of a formula changes at the zeros of its constraints and is invariant between the zeros. Thus, the idea is to collect all *symbolic zeros* $\text{zeros}(\varphi)$ of all constraints in some input formula φ. If all these constraints are weak, then a variable x_j is eliminated by plugging every zero and an arbitrarily small value $-\infty$ into the formula, i.e. $\exists x_j.\ \varphi$ is equivalent to $\varphi[-\infty//x_j] \vee \bigvee_{\xi\in\text{zeros}(\varphi)} \varphi[\xi//x_j]$. The formula $\varphi[t//x_j]$ encodes the semantics of substituting the term t for x_j into the formula φ (which is a disjunction of conjunctions). As we can pull existential quantifiers into disjunctions, we can iteratively eliminate multiple variables by handling each case separately.

The resulting algorithm for quantifier elimination is singly exponential; further optimizations ([26] even proposes to consider only lower or upper bounds for the test candidates) lead to a very similar procedure as the FMplex quantifier elimination: Substituting a test candidate into the formula is equivalent to computing the restricted projection w.r.t. a variable bound. However, our presentation allows to exploit the correspondence with the FM method.

Virtual substitution can also be adapted for SMT solving [3] to a depth-first search similar to FMplex. A conflict-driven search for virtual substitution on conjunctions of weak linear constraints has been introduced in [15], which tracks intermediate constraints as linear combinations of the input constraints similarly to FMplex. Their conflict analysis is a direct generalization of the global conflicts in FMplex and is thus slightly stronger than our notion of local conflicts. However, their method requires storing and maintaining a lemma database, while FMplex stores all the information for pruning the search tree locally. The approaches have strong similarities, although they originate from quite different methods. Further, our presentation shows the similarities to simplex, is easily adaptable for strict constraints, and naturally extensible to work incrementally.

5.3 Sample-Based Methods

There exist several depth-first search approaches, including McMillan et al. [22], Cotton [5] and Korovin et al. [16, 17], which maintain and adapt a concrete (partial) variable assignment. They share the advantage that combinations of constraints are only computed to guide the assignment away from an encountered conflict, thus avoiding many unnecessary combinations which FM would compute.

Similar to FMplex, these methods perform a search with branching, backtracking and learning from conflicting choices. However, they branch on variable assignments, with infinitely many possible choices in each step. Interestingly, the bounds learned from encountered conflicts implicitly partition the search space into a finite number of regions to be tried, similar to what FMplex does explicitly. In fact, we deem it possible that [16] or [17] try and exclude assignments from exactly the same regions that FMplex would visit (even in the same order). However, the sample-based perspective offers different possibilities for heuristic improvements than FMplex: choosing the next assigned value vs. choosing the next lower bound; deriving constant variable bounds vs. structural exploits using Farkas' Lemma; possibility of very quick solutions vs. more control and knowledge about the possible choices.

Moreover, these methods offer no straight-forward adaption for quantifier elimination, while FMplex does. However, [22] and [5] can handle not only conjunctions, but any quantifier-free LRA formula in conjunctive normal form.

6 Experimental Evaluation

We implemented several heuristic variants of the FMplex algorithm, as well as the generalized *simplex* and the *FM* methods as non-incremental DPLL(T) theory backends in our SMT-RAT solver [4] and compared their performance in the context of satisfiability checking. Using the transformation given in [24] and case splitting as in [2], we extended the method to also handle strict and not-equal-constraints.

The base version of FMplex (Algorithm 1a) was tested with two different heuristics for the choice of the eliminated variable and for the order in which the branches are checked. These choices may strongly influence the size of the explored search tree; in the best case, the very first path leads to a satisfiable leaf or to a global conflict.

Min-Fanout We greedily minimize the number of children: for any $A\boldsymbol{x} \leq \boldsymbol{b}$ and I, we choose $V \in$ $branch\_choices(A\boldsymbol{x} \leq \boldsymbol{b}, I)$ such that $|V|$ is minimal; in case that this minimum is 1, we prefer choices $V = \{(x_j, \bot)\}$ for a $j \in [n]$ over the other choices.

We prefer rows with a lower (earlier) backtrack level, motivated by finding a global conflict through trying positive linear combinations first. Moreover, if backtracking is used then we expect this heuristic to allow for backtracking further back on average.

Min-Column-Length A state-of-the-art heuristic for simplex in the context of SMT solving is the *minimum column length* [14]: we choose the variables for leaving and entering the non-basis such that the number of necessary row operations is minimized. We resemble this heuristic in FMplex as follows: we prefer choices $\{(x_j, \bot)\}$ and if there is no such j, we take the $j \in [n]$ with minimal $|I_j^-(A)| + |I_j^+(A)|$ and take the smallest choice between $I_j^-(A)$ and $I_j^+(A)$.

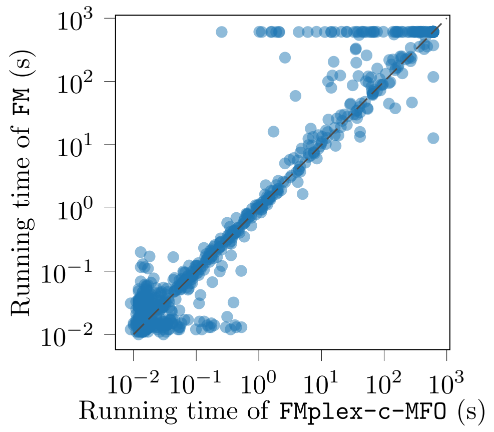
We first choose the rows which have the least non-zero coefficients (i.e. contain the least variables) to prefer sparse sub-problems. This can be understood as *Min-Row-Length*.

We consider the following solver variants: `FMplex-a-MFO` and `FMplex-a-MCL` implement Algorithm 1a with the Min-Fanout and the Min-Column-Length heuristic, respectively. `FMplex-a-Rand-1/2` denotes two variants of Algorithm 1a where all choices are taken pseudo-randomly with different seeds. `FMplex-b-MFO` implements Algorithm 1b and `FMplex-c-MFO` implements Algorithm 1c , both using the Min-Fanout heuristic. Our approach is also compared to non-incremental implementations `FM` and `Simplex`. The FMplex variants and `FM` always first employ Gaussian elimination to handle equalities.

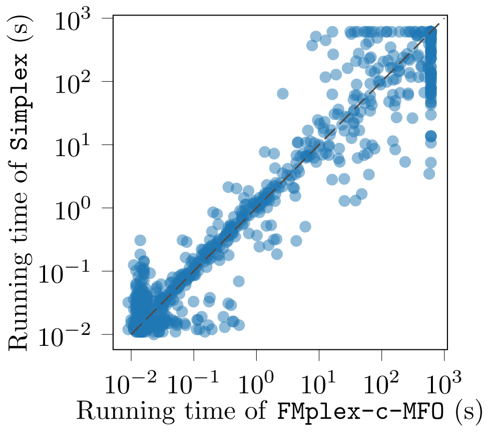
All solvers were tested on the SMT-LIB [1] benchmark set for QF_LRA containing 1753 formulas. As all evaluated solvers are non-incremental, we also generated conjunctions of constraints by solving each of these QF_LRA problems using a DPLL(T) SMT solver with an `FMplex-c-MFO` theory solver backend, and extracting all conjunctions passed to it. If the solver terminated within the time and memory limits, we sampled 10 satisfiable and 10 unsatisfiable conjunctions (or gathered all produced conjunctions if there were fewer than 10). This amounted to 3084 (777 sat, 2307 unsat) additional benchmarks. The experiments were conducted on identical machines with two Intel Xeon Platinum 8160 CPUs (2.1 GHz, 24 cores). For each formula, the time and memory were limited to 10 minutes and 5 GB.

	SMT-LIB					Conjunctions				
	solved	sat	unsat	TO	MO	solved	sat	unsat	TO	MO
`Simplex`	958	527	431	714	81	3084	777	2307	0	0
`FM`	860	461	399	577	316	2934	747	2187	107	43
`FMplex-a-MFO`	814	432	382	840	99	2962	743	2219	122	0
`FMplex-a-MCL`	820	435	385	830	103	2965	742	2223	119	0
`FMplex-a-Rand-1`	742	383	359	906	105	2806	668	2138	278	0
`FMplex-a-Rand-2`	743	383	360	905	105	2823	671	2152	261	0
`FMplex-b-MFO`	822	434	388	830	101	2988	744	2244	96	0
`FMplex-c-MFO`	920	499	421	733	100	3084	777	2307	0	0
`Virtual-Best`	982	532	450	651	120	3084	777	2307	0	0

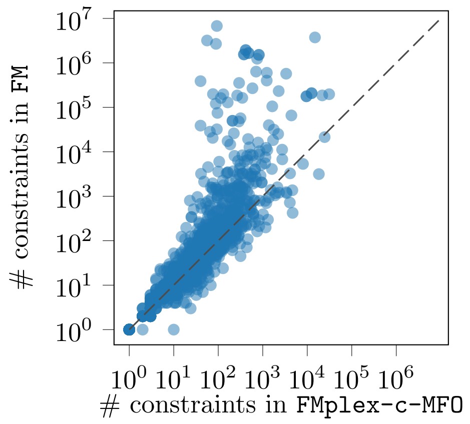
Table 1: Number of solved instances, timeouts (TO) and memory-outs (MO).



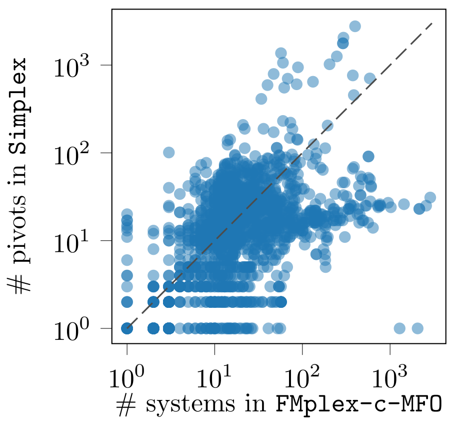
(a) Running times in seconds on the SMT-LIB benchmark set, `FMplex` vs `FM`.



(b) Running times in seconds on the SMT-LIB benchmark set, `FMplex` vs `Simplex`.



(c) Number of generated constraints on the conjunctive benchmark set.



(d) Number of visited non-bases (intermediate systems) on the conjunctive benchmark set.

Figure 3: Scatter plots: Each dot represents a single instance. In (a) and (b), instances at the very top or right exceeded the resource limit. Such instances are not considered in (c) and (d).

The results in Table 1 show that `Simplex` solved the most SMT-LIB instances, followed by our `FMplex-c-MFO` and then `FM`. Interestingly, `FM` solves fewer conjunctive instances than the base version of FMplex due to higher memory consumption (43 memory-outs for `FM`, while the others have none). We see that a reasonable variable heuristic makes a difference as `FMplex-a-Rand-*` perform much worse than `FMplex-a-MFO` and `FMplex-a-MCL`. However, between the latter two, there is no significant difference. While our first optimization used in `FMplex-b-MFO` has no big impact, the backtracking implemented in `FMplex-c-MFO` allows for solving more instances within the given resource limits.

The running times for each individual SMT-LIB instance depicted in Figures 3a and 3b reveal that `FM` and `FMplex-c-MFO` often behave similar, but `FM` fails on a number of larger instances. We suspect that the smaller intermediate systems of FMplex are a main factor here. While `Simplex` is often faster than `FMplex-c-MFO` and solves 61 SMT-LIB instances not solved by `FMplex-c-MFO`, it fails to solve 23 instances on which `FMplex-c-MFO` succeeds (Of these instances, `FM` solves 3 respectively 14 instances). Accordingly, the `Virtual-Best` of the tested solvers performs significantly better than just `Simplex`, indicating potential for a combination of `Simplex` and `FMplex-c-MFO`.

Figure 3c compares the number of constraints generated by `FM` and `FMplex-c-MFO` on the conjunctive inputs. Especially on larger instances, FMplex seems to be in the advantage. Motivated by Section 4.1, Figure 3d compares the number of `Simplex` pivots to the number of systems in `FMplex-c-MFO`. We see that neither is consistently lower than the other, though `Simplex` seems to be slightly superior. Due to the log-log scale, not shown are 1305 instances in which either measurement is 0 (920 instances for `Simplex`, 981 for `FMplex-c-MFO`).

The implementation and collected data are available at `https://doi.org/10.5281/zenodo.7755862`.

7 Conclusion

We introduced a novel method *FMplex* for quantifier elimination and satisfiability checking for conjunctions of linear real arithmetic constraints. Structural observations based on Farkas' Lemma and the Fundamental Theorem of Linear Programming allowed us to prune the elimination or the search tree. Although the new method is rooted in the FM method, it has strong similarities with both the virtual substitution method and the simplex method.

The experimental results in the context of SMT solving show that FMplex is faster than Fourier-Motzkin and, although simplex is able to solve more instances than FMplex, there is a good amount of instances which can be solved by FMplex but cannot be solved by simplex.

In future work, we aim to combine the structural savings of FMplex with the efficient heuristic of simplex, i.e. we transfer ideas from FMplex to simplex and vice-versa. Furthermore, we will investigate in tweaks and heuristics. For instance, we plan to adapt the perfect elimination ordering from [19] and work on an incremental adaption for SMT solving. Last but not least, we plan to increase the applicability of FMplex as a quantifier elimination procedure, including a different handling of strict inequalities, which is more similar to FM.

References

[1] Clark Barrett, Pascal Fontaine & Cesare Tinelli (2016): *The Satisfiability Modulo Theories Library (SMT-LIB).* `www.SMT-LIB.org`.

[2] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras & Cesare Tinelli (2006): *Splitting on Demand in SAT Modulo Theories*. In: *International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'06)*, Springer, pp. 512–526, doi:10.1007/11916277_35.

[3] Florian Corzilius & Erika Ábrahám (2011): *Virtual Substitution for SMT-Solving*. In: *International Symposium on Fundamentals of Computation Theory (FCT'11)*, Springer, pp. 360–371, doi:10.1007/978-3-642-22953-4_31.

[4] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp & Erika Ábrahám (2015): *SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving*. In: *International Conference on Theory and Applications of Satisfiability Testing (SAT'15)*, Springer, pp. 360–368, doi:10.1007/978-3-319-24318-4_26.

[5] Scott Cotton (2010): *Natural Domain SMT: A Preliminary Assessment*. In: *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'10)*, Springer, pp. 77–91, doi:10.1007/978-3-642-15297-9_8.

[6] George Bernard Dantzig (1998): *Linear Programming and Extensions*. 48, Princeton University Press, doi:10.1515/9781400884179.

[7] Bruno Dutertre & Leonardo De Moura (2006): *Integrating Simplex with DPLL(T)*. *Computer Science Laboratory, SRI International, Tech. Rep. SRI-CSL-06-01.*

[8] Julius Farkas (1902): *Theorie der einfachen Ungleichungen*. *Journal für die reine und angewandte Mathematik (Crelles Journal)* 1902(124), pp. 1–27, doi:10.1515/crll.1902.124.1.

[9] Jean Baptiste Joseph Fourier (1827): *Analyse des travaux de l'Académie Royale des Sciences pendant l'année 1824*. *Partie mathématique.*

[10] Jean-Louis Imbert (1990): *About Redundant Inequalities Generated by Fourier's Algorithm*. In: *International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA'90)*, Elsevier, pp. 117–127, doi:10.1016/B978-0-444-88771-9.50019-2.

[11] Jean-Louis Imbert (1993): *Fourier's Elimination: Which to Choose?* In: *International Conference on Principles and Practice of Constraint Programming (PPCP'93)*, 1, Citeseer, pp. 117–129.

[12] Rui-Juan Jing, Marc Moreno-Maza & Delaram Talaashrafi (2020): *Complexity Estimates for Fourier-Motzkin Elimination*. In: *Computer Algebra in Scientific Computing (CASC'20)*, Springer, pp. 282–306, doi:10.1007/978-3-030-60026-6_16.

[13] Leonid Genrikhovich Khachiyan (1980): *Polynomial Algorithms in Linear Programming*. *USSR Computational Mathematics and Mathematical Physics* 20(1), pp. 53–72, doi:10.1016/0041-5553(80)90061-0.

[14] Tim King, Clark Barrett & Bruno Dutertre (2013): *Simplex with Sum of Infeasibilities for SMT*. In: *Formal Methods in Computer-Aided Design (FMCAD'13)*, pp. 189–196, doi:10.1109/FMCAD.2013.6679409.

[15] Konstantin Korovin, Marek Kosta & Thomas Sturm (2014): *Towards Conflict-driven Learning for Virtual Substitution*. In: *International Workshop on Computer Algebra in Scientific Computing (CASC'14)*, Springer, pp. 256–270, doi:10.1007/978-3-319-10515-4_19.

[16] Konstantin Korovin, Nestan Tsiskaridze & Andrei Voronkov (2009): *Conflict Resolution*. In: *Principles and Practice of Constraint Programming (CP'09)*, Springer, pp. 509–523, doi:10.1007/978-3-642-04244-7_41.

[17] Konstantin Korovin & Andrei Voronkov (2011): *Solving Systems of Linear Inequalities by Bound Propagation*. In: *Conference on Automated Deduction (CADE'23)*, Springer, pp. 369–383, doi:10.1007/978-3-642-22438-6_28.

[18] Carlton E. Lemke (1954): *The Dual Method of Solving the Linear Programming Problem*. *Naval Research Logistics Quarterly* 1(1), pp. 36–47, doi:10.1002/nav.3800010107.

[19] Haokun Li, Bican Xia, Huiying Zhang & Tao Zheng (2021): *Choosing the Variable Ordering for Cylindrical Algebraic Decomposition via Exploiting Chordal Structure*. In: *International Symposium on Symbolic and Algebraic Computation (ISSAC'21)*, ACM, pp. 281–288, doi:10.1145/3452143.3465520.

[20] Rüdiger Loos & Volker Weispfenning (1993): *Applying Linear Quantifier Elimination*. *The Computer Journal* 36(5), pp. 450–462, doi:10.1093/comjnl/36.5.450.

[21] David G Luenberger, Yinyu Ye et al. (1984): *Linear and Nonlinear Programming*. 2nd edition, Springer, doi:10.1007/978-3-030-85450-8.

[22] Kenneth L. McMillan, Andreas Kuehlmann & Mooly Sagiv (2009): *Generalizing DPLL to Richer Logics*. In: *International Conference on Computer Aided Verification (CAV'09)*, Springer, pp. 462–476, doi:10.1007/978-3-642-02658-4_35.

[23] Theodore Samuel Motzkin (1936): *Beiträge zur Theorie der linearen Ungleichungen*. Azriel.

[24] Jasper Nalbach, Erika Ábrahám & Gereon Kremer (2021): *Extending the Fundamental Theorem of Linear Programming for Strict Inequalities*. In: *International Symposium on Symbolic and Algebraic Computation (ISSAC'21)*, ACM, pp. 313–320, doi:10.1145/3452143.3465538.

[25] Jasper Nalbach, Valentin Promies, Erika Ábrahám & Paul Kobialka (2023): *FMplex: A Novel Method for Solving Linear Real Arithmetic Problems*. arXiv:2309.03138.

[26] Tobias Nipkow (2008): *Linear Quantifier Elimination*. In: *Internation Joint Conference on Automated Reasoning (IJCAR'08)*, Springer, pp. 18–33, doi:10.1007/978-3-540-71070-7_3.

[27] Volker Weispfenning (1997): *Quantifier Elimination for Real Algebra—the Quadratic Case and Beyond*. *Applicable Algebra in Engineering, Communication and Computing* 8(2), pp. 85–101, doi:10.1007/s002000050055.

Handling of Past and Future with Phenesthe+

Manolis Pitsikalis Alexei Lisitsa
Patrick Totzke

Department of Computer Science, University of Liverpool, United Kingdom

{e.pitsikalis, a.lisitsa, totzke}@liverpool.ac.uk

Writing temporal logic formulae for properties that combine instantaneous events with overlapping temporal phenomena of some duration is difficult in classical temporal logics. To address this issue, in previous work we introduced a new temporal logic with intuitive temporal modalities specifically tailored for the representation of both instantaneous and durative phenomena. We also provided an implementation of a complex event processing system, Phenesthe, based on this logic, that has been applied and tested on a real maritime surveillance scenario.

In this work, we extend our temporal logic with two extra modalities to increase its expressive power for handling future formulae. We compare the expressive power of different fragments of our logic with Linear Temporal Logic and dyadic first-order logic. Furthermore, we define correctness criteria for stream processors that use our language. Last but not least, we evaluate empirically the performance of Phenesthe+, our extended implementation, and show that the increased expressive power does not affect efficiency significantly.

1 Introduction

Temporal logics are widely used in many domains as they allow the formalisation of time dependent properties. For example in philosophy temporal logics can be used to reason about issues involving the temporal domain [28], in computer science and specifically, in monitoring, temporal logics are used to specify and monitor specific properties of a system [21, 6], in complex event processing or recognition [12, 3, 7]—which is the focus of this work—temporal logics are used for specifying temporal phenomena and detecting them in streams of information. Naturally, each temporal logic comes with its own focus and limitations.

The starting point for many monitoring systems is Linear Temporal Logic (LTL) [24], where formulae are interpreted over single event sequences. This makes it difficult to incorporate concurrent activities such as, for instance, those carried out by a chef following some recipe to create a meal (see Figure 1). They may prepare multiples dishes of the same course in parallel, however the preparation of each dish happens on different overlapping intervals. This is difficult to formalise with logics that talk about single traces of events. Temporal logics that allow the representation of concurrent activities directly are those of Halpern and Shoham (HS) [16] and Allen's Algebra [2]. Both logics are interpreted over sets of (possibly overlapping, discrete) time intervals, and model instantaneous events via point intervals ([t,t]).

In previous work [22], we introduced a temporal logic similar to those of Halpern/Shoham and Allen, which is interpreted over separate (sets of) time intervals and instantaneous events, which was specifically designed for a maritime application domain. It allows to easily specify concurrent activities and related start and endpoints. Deliberately absent in our logics (and related prior ones) are explicit negations/complements of formulae that hold on overlapping intervals. We implemented an event processing

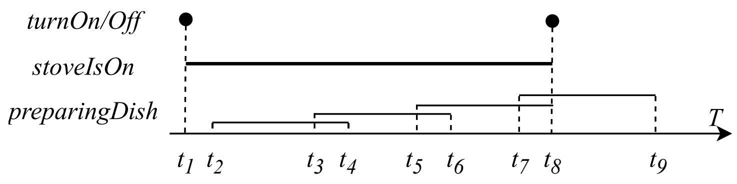


Figure 1: Example of instantaneous and durative temporal phenomena. *turnOn/Off* is instantaneous and true when the stove is turned on/off, *stoveIsOn* is durative and holds when the stove is on, and finally *preparingDish* is also durative and holds when a dish is being prepared.

system[1] and evaluated it on a real maritime monitoring scenario [23]—whereby maritime experts authored maritime phenomena of interest—proving that our system is capable of producing phenomena detections in real time.

In this work, we extend our language with two extra temporal modalities: minimal range ($\looparrowright$), and interval filtering (filter). The minimal range operator allows to capture durative temporal phenomena that start at the latest occurrence of a starting condition before the stopping condition, for example, the last working period of machine since someone operated it until it broke down. Filtering is very important for specifying the duration constraints of a temporal phenomenon, e.g., a steak is cooked rare if its on a hot pan for approximately 90 seconds on each side. Both examples are impossible to formalise in the original version of our temporal logic. It is evident that $\looparrowright$ has similar semantics to the 'until' operator of LTL. However, formulae that utilise 'until' are true on instants of time, while formulae that utilise '$\looparrowright$' are true on intervals. While in terms of expressive power the two operators are similar, $\looparrowright$, in practice, allows efficient computations and more concise formulae. Similarly, in the case of filtering, writing an LTL formula for filtering periods based on some fixed threshold is possible, however the formula is not trivial and its length depends the threshold. Comparing LTL and our temporal logic, we show that the fragment of our original language that is comparable with LTL was at most as expressive as the pure past fragment of LTL, while the same fragment but with the addition of the '$\looparrowright$' and 'filter' operators has equal expressive power to LTL. Concerning our full temporal logic, a comparison with LTL is impossible since the structures on which their respective semantics are based, are incomparable, however we show that our language is expressible in Dyadic First Order logic (DFO) and is strictly less expressive. As expected, including temporal modalities that involve the future requires additional steps for complex event processing. In order to guarantee that our extended implementation, Phenesthe+, is correct, inspired from runtime monitoring and verification [1, 5], we define criteria for *proper* stream processors of our language and discuss how Phenesthe+ conforms to them. Finally, we illustrate through experimental evaluation that the efficiency of Phenesthe+, is not significantly compromised. Therefore the contributions of this paper are:

- We extend the expressiveness of our language for representing "look ahead happenings",
- We formally study the expressive power of the temporal logic introduced in [22] and its extension,
- We define criteria for *proper* stream processors utilising our temporal logic,
- We showcase that our stream processing engine is capable of performing real-time complex event processing by adopting a maritime surveillance use-case.

The paper is organised as follows. First in Section 2 we describe our temporal logic. Next, in Section 3 we illustrate through examples inspired by the maritime domain the usage of the new temporal modalities.

[1]`https://manospits.github.io/Phenesthe/`

In Section 4 we study the expressive power of our temporal logic. Then, in Section 5 we describe the requirements a stream processor should satisfy for processing formulae of our language, while in Section 6 we empirically evaluate Phenesthe+. Finally, in Section 7 we present work related to ours, summarise, and discuss further directions.

2 The Language of Phenesthe

The key components of our language are instantaneous events, durative disjoint states and durative, possibly non disjoint, dynamic temporal phenomena. In what follows, 'temporal phenomena' includes all of the three aforementioned categories.

Syntax. Formally, our Temporal Phenomena Definition Language (*TPhL*) is described by the triplet $\langle \mathcal{P}^{esd}, L, \Phi \rangle$, where $\mathcal{P}^{esd}$ is a predicate set defined by the union of the event, state or dynamic temporal phenomenon predicates sets (in symbols $\mathcal{P}^{e/s/d}$ resp.); L is a set defined by the union of the set of the logical connectives $\{\wedge, \vee, \neg, \in\}$, the set of temporal operators, $\{\rightarrowtail, \looparrowright, \sqcup, \sqcap, \setminus, \mathsf{filter}_{\square}\}$ where the '$\square$' symbol may be one of the following symbols $\{<, \geq, =\}$, the set of temporal relations $\{$before, meets, overlaps, finishes, starts, equals, contains$\}$ and finally the set of the $\{\mathsf{start}, \mathsf{end}\}$ operators; Φ is the set of formulae defined by the union of the formulae sets $\Phi^{\boldsymbol{\cdot}}$, Φ^{-} and $\Phi^{=}$. We assume that the set of predicate symbols includes those with atemporal and fixed semantics, such as arithmetic comparison operators etc., however for simplification reasons in what follows we omit their presentation. Formulae of $\Phi^{\boldsymbol{\cdot}}$ describe instantaneous temporal phenomena, and formulae of Φ^{-} describe durative temporal phenomena that hold (are true) in disjoint maximal intervals, finally formulae of $\Phi^{=}$ describe durative temporal phenomena that may hold in non-disjoint intervals. Figure 1 shows an example of an event (*turnOn*/*Off*), a state (*stoveIsOn*) and a dynamic temporal phenomenon (*preparingDish*).

Therefore given a set of event, state and dynamic phenomena predicates $\mathcal{P}^{esd}$ the formulae of *TPhL* are defined as follows:

$$
\begin{aligned}
\phi &:= \phi^{\boldsymbol{\cdot}} \mid \phi^{-} \mid \phi^{=} \\
\phi^{\boldsymbol{\cdot}} &:= P^{e}(a_1,...,a_k) \mid \neg\phi^{\boldsymbol{\cdot}} \mid \phi^{\boldsymbol{\cdot}}\ [\wedge,\vee]\ \phi^{\boldsymbol{\cdot}} \mid \mathsf{start}(\phi^{-}) \mid \mathsf{end}(\phi^{-}) \mid \phi^{\boldsymbol{\cdot}} \in \phi^{-} \\
\phi^{-} &:= P^{s}(a_1,...,a_k) \mid \phi^{\boldsymbol{\cdot}}\ [\rightarrowtail,\looparrowright]\ \phi^{\boldsymbol{\cdot}} \mid \phi^{-}\ [\sqcup,\sqcap,\setminus]\ \phi^{-} \mid \phi^{-}\mathsf{filter}\ n \text{ (where } n \in \mathbb{N}\cup\{\infty\}) \\
\phi^{=} &:= P^{d}(a_1,...,a_k) \mid [\phi^{-},\phi^{=}]\ [\mathsf{meets},\mathsf{overlaps},\mathsf{equals}]\ [\phi^{-},\phi^{=}] \\
&\qquad \mid [\phi^{\boldsymbol{\cdot}},\phi^{-},\phi^{=}]\ [\mathsf{starts},\mathsf{finishes}]\ [\phi^{-},\phi^{=}] \\
&\qquad \mid [\phi^{-},\phi^{=}]\ \mathsf{contains}\ [\phi^{\boldsymbol{\cdot}},\phi^{-},\phi^{=}] \\
&\qquad \mid [\phi^{\boldsymbol{\cdot}},\phi^{-},\phi^{=}]\ \mathsf{before}\ [\phi^{\boldsymbol{\cdot}},\phi^{-},\phi^{=}]
\end{aligned}
$$

where $a_1,...,a_k$ correspond to terms denoting atemporal properties.

Semantics. We assume time is discrete and represented by the natural numbers $T = \mathbb{N}$ ordered via the '$<$' relation. In what follows we assume that for all models discussed in this paper time is represented by $\mathbb{N}$. For the formulae sets $\Phi^{\boldsymbol{\cdot}}, \Phi^{-}$ and $\Phi^{=}$ we define the model $\mathcal{M} = \langle T, I, <, V^{\boldsymbol{\cdot}}, V^{-}, V^{=} \rangle$ where $V^{\boldsymbol{\cdot}} : \mathcal{P}^{e} \to 2^{T}$, $V^{-} : \mathcal{P}^{s} \to 2^{I}$, $V^{=} : \mathcal{P}^{d} \to 2^{I}$ are valuations, and $I = \{[ts,te] : ts < te \text{ and } ts,te \in T\} \cup \{[ts,\infty) : ts \in T\}$ is the set of time intervals of T. Intervals of the form $[ts,\infty)$ denote that a phenomenon started being true at ts, and continues being true forever. Intervals of the form $[ts,te]$ denote that a phenomenon started being true at ts and stopped being true at te. In what follows, we will use the abbreviated version for bounded quantifiers, i.e., $\forall^{<z_2}_{>z_1} x(...)$ denotes $\forall x\ (x > z_1 \wedge x < z_2) \to (...)$, and $\exists^{<z_2}_{>z_1} x(...)$ denotes $\exists x\ (x > z_1 \wedge x < z_2) \wedge (...)$.

Given a model $\mathcal{M}$, the validity of a formula $\phi \in \Phi^{\bullet}$ at a timepoint $t \in T$ (in symbols $\mathcal{M}, t \models \phi$) is determined by the rules below, starting with the boolean connectives.

- $\mathcal{M}, t \models P^e(a_1, ..., a_n)$ iff $t \in V^{\bullet}(P^e(a_1, ..., a_n))$.
- $\mathcal{M}, t \models \neg\phi$ iff $\mathcal{M}, t \not\models \phi$.
- $\mathcal{M}, t \models \phi[\wedge, \vee]\psi$ iff $\mathcal{M}, t \models \phi$ [and, or] $\mathcal{M}, t \models \psi$.

Next we define the semantics for $\mathsf{start}, \mathsf{end}$ and $\in$ which allow interaction between formulae of $\Phi^{\bullet}$ and Φ^{-} via the starting, ending, and intermediate points of intervals at which Φ^{-} formulae hold.

- $\mathcal{M}, t \models \mathsf{start}(\phi)$ iff $\exists te.\ \mathcal{M}, [t, te] \models \phi$ or $\mathcal{M}, [t, \infty) \models \phi$, where $\mathcal{M}, [t, te] \models \phi$ denotes the validity of a formula $\phi \in \Phi^{-}$ at an interval $[t, te]$ as defined below.
- $\mathcal{M}, t \models \mathsf{end}(\phi)$ iff $\exists ts.\ \mathcal{M}, [ts, t] \models \phi$.
- $\mathcal{M}, t \models \phi \in \psi$ iff $\mathcal{M}, t \models \phi$ and $\exists^{\leq t} ts. \exists_{\geq t} te.\ \mathcal{M}, [ts, te] \models \psi$.

Given a model $\mathcal{M}$, the validity of a formula $\phi \in \Phi^{-}$ at a time interval $i = [ts, te] \in I$ (in symbols $\mathcal{M}, [ts, te] \models \phi$) is defined as follows. We start with the $\looparrowright$ and $\rightarrowtail$ operators which allow specifying minimal or maximal intervals between instants where formulae of $\Phi^{\bullet}$ are true.

- $\mathcal{M}, i \models P^s(a_1, ..., a_n)$ iff $i \in V^{-}(P^s(a_1, ..., a_n))$.
- $\mathcal{M}, [ts, te] \models \phi \looparrowright \psi$ iff $\mathcal{M}, ts \models \phi$ and $\mathcal{M}, te \models \psi \wedge \neg\phi$ and $\forall^{<te}_{>ts} t. \big[\mathcal{M}, t \not\models \phi$ and $\mathcal{M}, t \not\models \psi \wedge \neg\phi\big]$. Therefore, $\phi \looparrowright \psi$ holds for the intervals that start at the latest instant ts at which ϕ is true and end at first instant te after ts where $\psi \wedge \neg\phi$ is true.
- $\mathcal{M}, [ts, te] \models \phi \rightarrowtail \psi$ iff $\mathcal{M}, ts \models \phi$ and $\mathcal{M}, te \models \psi \wedge \neg\phi$ and $\forall^{<te}_{>ts} t.\ \mathcal{M}, t \not\models \psi \wedge \neg\phi$ and $\forall^{<ts} ts'.$ $\mathcal{M}, ts' \models \phi \rightarrow \exists^{<ts}_{>ts'} te'.\ \mathcal{M}, te' \models \psi \wedge \neg\phi$.

  Essentially, $\phi \rightarrowtail \psi$ holds for the disjoint maximal intervals that start at the earliest instant ts where ϕ is true and end at the earliest instant te where ψ is true and ϕ is false.
- $\mathcal{M}, [ts, \infty) \models \phi \rightarrowtail \psi$ iff $\mathcal{M}, ts \models \phi$ and $\forall_{>ts} t.\ \mathcal{M}, t \not\models \psi \wedge \neg\phi$ and $\forall^{<ts} ts'.\ \mathcal{M}, ts' \models \phi \rightarrow \exists^{<ts}_{>ts'} te'.$ $\mathcal{M}, te \models \psi \wedge \neg\phi$.

  Therefore a formula $\phi \rightarrowtail \psi$ may hold indefinitely if there does not exist an instant after ts at which $\psi \wedge \neg\phi$ is satisfied. For simplification reasons in the semantics below we omit intervals open at the right to infinity since they can be treated in a similar manner.

We continue with the definition of semantics for $\sqcup, \sqcap$ and $\setminus$ which correspond to the usual set operations but for time intervals.

- $\mathcal{M}, [ts, te] \models \phi \sqcup \psi$ iff[2]
  - exists a sequence of length $k > 1$ of intervals $i_1, ..., i_k \in I$ where $i_k = [ts_k, te_k]$, $ts = ts_1$ and $te = te_k$ such that:
    1. $\forall \alpha \in [1, k-1]$: $te_\alpha \in i_{\alpha+1}$, $ts_\alpha < ts_{\alpha+1}$ and $te_\alpha < te_{\alpha+1}$,
    2. $\forall \beta \in [1, k]$: $\mathcal{M}, [ts_\beta, te_\beta] \models \phi$ or $\mathcal{M}, [ts_\beta, te_\beta] \models \psi$, and
    3. $\nexists i_\gamma = [ts_\gamma, te_\gamma] \in I - \{i_1, ..., i_k\}$ where $\mathcal{M}, [ts_\gamma, te_\gamma] \models \phi$ or $\mathcal{M}, [ts_\gamma, te_\gamma] \models \psi$ and $ts_1 \in i_\gamma$ or $te_k \in i_\gamma$
  - or, $\mathcal{M}, [ts, te] \models \phi$ or $\mathcal{M}, [ts, te] \models \psi$ and $\nexists i_\gamma = [ts_\gamma, te_\gamma] \in I - \{[ts, te]\}$ where $\mathcal{M}, [ts_\gamma, te_\gamma] \models \phi$ or $\mathcal{M}, [ts_\gamma, te_\gamma] \models \psi$ and $ts \in i_\gamma$ or $te \in i_\gamma$.

[2]A first order definition of the semantics of $\sqcup$ is possible but more lengthy.

For a sequence of intervals, conditions (1-2) ensure that intervals, at which ϕ or ψ are valid, overlap or touch will coalesce, while condition (3) ensures that the resulting interval is maximal. In the case of a single interval, the conditions ensure that at the interval $[ts,te]$ ϕ or ψ is valid, and that $[ts,te]$ is maximal. In simple terms, the temporal union $\phi \sqcup \psi$ holds for the intervals where at least one of ϕ or ψ hold. The above definition of temporal union follows the definitions of temporal coalescing presented in [8, 14].

- $\mathcal{M},[ts,te] \models \phi \setminus \psi$ iff $\exists[ts',te'] \in I$ where $\mathcal{M},[ts',te'] \models \phi$, $[ts,te] \subseteq [ts',te']$ (i.e., $[ts,te]$ subinterval of $[ts',te']$), $\forall[ts_\psi,te_\psi] \in I$ where $\mathcal{M},[ts_\psi,te_\psi] \models \psi$, $[ts,te] \cap [ts_\psi,te_\psi] = \varnothing$ and finally $[ts,te]$ is maximal. In plain language, the temporal difference of formulae ϕ,ψ holds for the maximal subintervals of the intervals at which ϕ holds but ψ doesn't hold.
- $\mathcal{M},[ts,te] \models \phi \sqcap \psi$ iff $\exists[ts_\phi,te_\phi],[ts_\psi,te_\psi] \in I$ where $\mathcal{M},[ts_\phi,te_\phi] \models \phi$, $\mathcal{M},[ts_\psi,te_\psi] \models \psi$ and $\exists[ts,te] \in I$ where $[ts,te] \subseteq [ts_\phi,te_\phi]$, $[ts,te] \subseteq [ts_\psi,te_\psi]$ and $[ts,te]$ is maximal. In other words, the temporal intersection of two formulae of Φ^- holds for the intervals at which both formulae hold.

We finish the semantics for formulae of Φ^-, with the semantics of the filter operator, which allows specifying constraints on the length of intervals at which formulae of Φ^- hold.

- $\mathcal{M},[ts,te] \models \phi \ \mathsf{filter}_{\{<,\geq,=\}}\ n$ iff $\mathcal{M},[ts,te] \models \phi$ and $te - ts\ \{<,\geq,=\}\ n$.

Due to space limitations and for this part only we will adopt point intervals to refer to instants. This will allow us to define the semantics for formulae of $\Phi^=$ without specifying different rules for involved sub-formulae of $\Phi^\bullet$. In other words given a $\phi \in \Phi^\bullet$ we will denote the satisfaction relation $\mathcal{M},t \models \phi$ as $\mathcal{M},[t,t] \models \phi$. Given a model $\mathcal{M}$, the validity of a formula $\phi \in \Phi^=$ at a time interval $[ts,te] \in I$ (in symbols $\mathcal{M},[ts,te] \models \phi$) is defined as follows:

- $\mathcal{M},[ts,te] \models P^d(a_1,...,a_n)$ iff $[ts,te] \in V^=(P^d(a_1,...,a_n))$.
- $\mathcal{M},[ts,te] \models \phi$ before ψ iff $\exists te'.\exists_{>te'} ts'.\big[\mathcal{M},[ts,te'] \models \phi$ and $\mathcal{M},[ts',te] \models \psi$ and $\forall ts''.\forall^{<ts'}_{>te'} te''.$ $\mathcal{M},[ts'',te''] \not\models \phi$ and $\forall^{<ts'}_{>te'} ts''.\forall te''.\mathcal{M},[ts'',te''] \not\models \psi\big]$. In our approach the 'before' relation holds only for intervals where the pair of instants or intervals at which the participating formulae are true or hold, are contiguous. For example, for the intervals $[1,2]$, $[1,3]$ and $[5,6]$ only $[1,3]$ is before $[5,6]$. We chose to limit the intervals satisfying the before relation, as in practice it is usually the case that the interval directly before another one is required for specifying a dynamic phenomenon.
- $\mathcal{M},[ts,te] \models \phi$ meets ψ iff $\exists t.\mathcal{M},[ts,t] \models \phi$ and $\mathcal{M},[t,te] \models \psi$.
- $\mathcal{M},[ts,te] \models \phi$ overlaps ψ iff $\exists^{<te}_{>ts} ts'.\exists^{<te}_{>ts'} te'.\big[\mathcal{M},[ts,te'] \models \phi$ and $\mathcal{M},[ts',te] \models \psi\big]$.
- $\mathcal{M},[ts,te] \models \phi$ finishes ψ iff $\exists^{\leq te}_{>ts} ts'.\big[\mathcal{M},[ts',te] \models \phi$ and $\mathcal{M},[ts,te] \models \psi\big]$.
- $\mathcal{M},[ts,te] \models \phi$ starts ψ iff $\exists^{<te}_{\geq ts} te'.\big[\mathcal{M},[ts,te'] \models \phi$ and $\mathcal{M},[ts,te] \models \psi\big]$.
- $\mathcal{M},[ts,te] \models \phi$ equals ψ iff $\mathcal{M},[ts,te] \models \phi$ and $\mathcal{M},[ts,te] \models \psi$.
- $\mathcal{M},[ts,te] \models \phi$ contains ψ iff $\mathcal{M},[ts,te] \models \phi$ and $\exists_{>ts} ts'.\exists^{<te} te'.\ \mathcal{M},[ts',te'] \models \psi$.

3 Examples of maritime properties expressed in *TPhL*

We demonstrate the usability of *TPhL* and the new temporal modalities by adopting a maritime monitoring scenario. When it comes to maritime surveillance there are several resources available; for example the Automatic Identification System (AIS) allows the transmission of timestamped positional and ancillary data from vessels, maritime areas in the form of polygons can be used for producing vessel-area relations and so on. Similar to [22], we assume the input consists of AIS messages along with spatial events relating vessels to areas of interest e.g., port areas, fishing areas and so on. Therefore our task here involves detecting maritime phenomena of interest i.e., the instants, time periods at which they are true over a maritime input stream. Below we formalise some maritime temporal phenomena[3] that utilise the new temporal modalities ($\looparrowright$ and filter).

Fishing warning. Illegal fishing is a very important issue. Vessels engaged in illegal fishing typically declare fake ship-types. Consider the formalisation below for detecting suspicious stops in fishing areas.

$$\begin{aligned}&\mathsf{state\_phenomenon}\, \mathit{fishing\_warning}(V,F):\\ &\quad ((\mathit{in\_fishing\_area}(V,F) \wedge \neg \mathit{vessel\_type}(V,\mathit{fishing})) \sqcap \mathit{stopped}(V))\ \mathsf{filter}_{\geq}600.\end{aligned}$$

$\mathsf{state\_phenomenon}$ is a keyword for declaring the phenomenon type, *in_fishing_area* is a user defined state that holds for the time periods a vessel V is within a fishing area F, while *stopped* is a state that holds for the time periods a vessel is stopped. Finally, $\mathit{vessel\_type}(V,T)$ is an atemporal predicate that is true when vessel V has type T. Therefore, a vessel performs a *fishing_warning*, if it is not a fishing vessel, and it is stopped within a fishing area for a period longer that 10 min (600 sec). Here, filtering is used for minimising false detections occurring from AIS errors (e.g., zero speed) or normal activities.

Port waiting time. Monitoring the waiting time of vessels since they entered a port and until they get moored is highly useful for various operational and logistical reasons (e.g., efficient planning of resources). However, some vessels may enter and leave a port without mooring—due to weather conditions for example. We formalise port waiting time below.

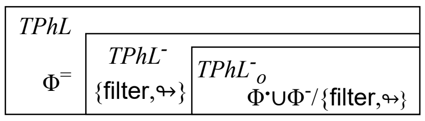
$$\begin{aligned}&\mathsf{state\_phenomenon}\ \mathit{waiting\_time}(V,P):\\ &\quad \mathsf{start}(\mathit{in\_port}(V,P)) \looparrowright \mathsf{start}(\mathit{moored}(V,P)).\end{aligned}$$

in_port is a state that holds when a vessel is in a port, while *moored* is a state that holds when a vessel is moored at a port. Note that the left and right arguments of $\looparrowright$ are formulae of $\Phi^{\bullet}$, therefore if these formulae were used in other definitions we could have defined corresponding events. Consequently, the *waiting_time* state holds for the minimal periods between the time a vessel enters a port and the time the vessel starts being moored. Here we are interested in the minimal period, as we want to detect only the cases where a vessel entered a port and got moored.

4 Expressiveness

In this section we study the expressive power of our language. We consider three syntactic fragments of *TPhL*. The first one, denoted as TPhL_o^-, corresponds to the original version of the language (w/o $\looparrowright, \mathsf{filter}$) and excluding formulae of $\Phi^{=}$ (recall that $\Phi^{=}$ formulae hold on possibly non-disjoint intervals). The second is TPhL^-, which is the same as TPhL_o^- but includes $\looparrowright$ and filter, while the third, *TPhL* corresponds to the complete language. Figure 2 (left) illustrates the syntactic relation between *TPhL*, TPhL^- and TPhL_o^-. In more detail, we will show that TPhL_o^- is equally expressive as pure past *LTL*, TPhL^- has equal expressive power to *LTL*, and finally *TPhL* is strictly less expressive than *DFO*. The relations in terms of expressive power between the different language fragments are illustrated in Figure 2 (right).

[3]The complete set of definitions is available in our online repository `https://github.com/manospits/Phenesthe/tree/future`.



$$\begin{tikzcd}
TPhL^- \arrow[r, phantom, "="] \arrow[d] & LTL[\texttt{XU},\texttt{YS}] \arrow[d] & DFO \arrow[d] \\
TPhL^-_o \arrow[r, equal] & LTL[\texttt{YS}] & TPhL
\end{tikzcd}$$

Figure 2: Syntactic relation between the fragments of *TPhL* (left). Expressive relations between different fragments of *TPhL*, LTL and *DFO* (right). A fragment A is strictly more expressive from a fragment B if they are connected via $A \rightarrow B$. Double lined edges denote equal expressive power.

4.1 Preliminaries

Before we continue with our analysis, as a reminder we present the syntax of LTL with past, and First Order Monadic Logic of Order (*FOMLO*).

LTL. The formulae of $LTL[\texttt{XU},\texttt{YS}]$, given a set of propositions $\mathcal{P}$ are defined as follows:

$$\phi ::= \bot \mid p \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \texttt{X}\phi_1 \mid \phi_1 \;\texttt{U}\; \phi_2 \mid \texttt{Y}\phi_1 \mid \phi_1 \;\texttt{S}\; \phi_2;$$

where X, and U stand for the next and until modalities, while Y, and S stand for previous and since. The formulae of $LTL[\texttt{XU},\texttt{YS}]$ are interpreted over a discrete, linear model of time, formally represented as $\mathcal{M}_{TL} = \langle T, <, V^{TL}\rangle$, where T is equal to $\mathbb{N}$, $<$ is the linear order and $V^{TL} : \mathcal{P} \rightarrow 2^T$ is the interpretation function, mapping each proposition to a set of time instants. The satisfaction relation, i.e., that a formula ϕ is true at t, is defined as $\mathcal{M}_{TL}, t \models \phi$. The semantics of *LTL* are defined as usual; more specifically in what follows we assume the reflexive[4] semantics of S and U. We denote the pure past fragment of *LTL* i.e., *LTL* without X and U as $LTL[\texttt{YS}]$.

FOMLO. Given a countable set of variables $x, y, z, ...$, the formulae of *FOMLO* over a set of unary predicate symbols Σ are defined a follows:

$$\begin{gathered} atomic ::= x < y \mid x = y \mid P(x) \text{ (where } P \in \Sigma) \\ \phi ::= atomic \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \exists x.\ \phi_1 \mid \forall x.\ \phi_1 \end{gathered}$$

We interpret *FOMLO* formulae over structures of the form $\mathcal{M}_{FO}\langle T, <, V^{FO}\rangle$, where T is equal to $\mathbb{N}$, '$<$' is the linear order while $V^{FO} : \Sigma \rightarrow 2^T$ is the interpretation of Σ. $\mathcal{M}_{FO}, t_1, t_2, \cdots, t_n \models \phi(x_1, x_2, \cdots, x_n)$ denotes the satisfaction of a formula ϕ with free variables $x_1, x_2, \cdots, x_n$ when they are interpreted as elements t_i of $\mathcal{M}_{FO}$. The semantics of the formulae are defined as usual (see for example [25]). We also define the $FOMLO^-$ fragment of *FOMLO*. Syntactically a formula with one free variable $\phi(x)$, is a formula of the fragment if any bounded variable in the negated normal form of $\phi(x)$ is bounded to be $\leq x$. Semantically, this means that for all models a formula $\phi(x)$ of $FOMLO^-$ satisfies:

$$\forall t\ \mathcal{M}_{FO}, t \models \phi(x) \leftrightarrow \mathcal{M}_{FO}[0,t], t \models \phi(x) \tag{1}$$

where $M_{FO}[0,t]$ is a finite model starting from 0 and ending up to position t inclusive. Intuitively, formulae of $FOMLO^-$ can talk only about the past and the present. In what follows, given a set of propositions $\mathcal{P} = \{p_1, ..., p_k\}$ and a set of predicate symbols $\Sigma = \{p_1(x), ..., p_k(x)\}$ a FOMLO model $\mathcal{M}_{FO}$ is *faithful* to $\mathcal{M}_{TL}$ iff $\forall_{\geq 1}^{\leq k} i. V^{FO}(p_i(x)) = V^{TL}(p_i)$.

DFO. Finally, on our expressiveness analysis we will also consider *DFO*, which in contrast to *FOMLO*, uses dyadic predicate symbols e.g., $p(x,y)$.

[4]As we work with discrete linear orders, this choice makes no difference.

4.2 $TPhL_o^-$ and Pure Past *LTL*

In this section we will show that $TPhL_o^-$ is expressively equal to the pure past fragment of *LTL*, i.e., $LTL[\mathtt{YS}]$. The $TPhL_o^-$ fragment is described by the triplet $\langle \mathcal{P}^{es}, L_o^-, \Phi \rangle$, where $\mathcal{P}^{es}$ is a set defined by the union of event and state predicate sets ($\mathcal{P}^{e/s}$); L_o^- is a set defined by the union of the set of the logical connectives $\{\wedge, \vee, \neg, \in\}$ and the set of temporal operators $\{\rightarrowtail, \sqcup, \sqcap, \backslash\}$. Formulae $TPhL_o^-$ are evaluated over $\mathcal{M}^- = \langle T, I, <, V^\bullet, V^- \rangle$ models which are defined in a similar manner to the models presented in Section 2.

Given a finite set of event propositions[5] $\mathcal{P}^e = \{e_1, ..., e_k\}$, and a finite set of state propositions $\mathcal{P}^s = \{s_1, ..., s_k\}$ we say the $FOMLO^-$ model $M_{FO} = \langle T, <, V^{FO} \rangle$ is *faithful* to the model $\mathcal{M}^- = \langle T, I, <, V^\bullet, V^- \rangle$ of $TPhL_o^-$ if it has the following properties:

- for any proposition e in $\mathcal{P}^e$, $V^\bullet(e) = V^{FO}(e_t)$, and
- for any proposition s in $\mathcal{P}^s$, $V^-(s) = \rho(V^{FO}(s_+), V^{FO}(s_\in), V^{FO}(s_-))$

where e_t corresponds to the monadic predicate $e_t(x)$ and the triplet $(s_+, s_\in, s_-)$ corresponds to the monadic predicates $s_+(x)$, $s_\in(x)$, $s_+(x)$ which are true on instants corresponding to the start, intermediate, and end of an interval respectively at which s is true. $\rho : 2^T \times 2^T \times 2^T \to 2^I$ is a partial mapping from three set of points to a set of intervals of I. Given three sets S, B and E, corresponding to starting, intermediate and ending points resp., ρ is defined as follows:

$$\rho(S,B,E) = \begin{array}{l} \big\{[ts,te] : ts < te \wedge ts \in S \wedge te \in E \wedge \forall_{>ts}^{<ts} t.(t \in B \wedge t \notin S \wedge t \notin E)\big\} \\ \cup \big\{[ts,\infty) : ts \in S \wedge \nexists_{>ts} te.\ te \in E \wedge \forall_{>ts} t.\ t \in B\big\} \end{array}$$

For all $i \in I$ where a formula $\phi \in \Phi^-$ is true ρ is bijective (recall that $\phi \in \Phi^-$ formulae always hold on disjoint intervals). For our expressiveness study, we will use the following theorem.

Theorem 1 (Gabbay et al. 1980 [15])**.** *For every formula of $FOMLO^-$ $\phi(x)$ we can find an $LTL[\mathtt{YS}]$ formula θ such that $\forall t.\mathcal{M}_{TL}, t \models \phi \leftrightarrow \mathcal{M}_{FO}, t \models \theta$ for all $\mathcal{M}_{FO}$ and their faithful models $\mathcal{M}_{TL}$.*

Proof. Dual proof of Theorem 2.2 in [15]. □

Theorem 2. *For every formula ϕ of $TPhL_o^-$, for all $TPhL_o^-$ models $\mathcal{M}^-$ and their faithful $FOMLO^-$ models $\mathcal{M}_{FO}$:*

1. *if $\phi \in \Phi^\bullet$, there exists a formula $\phi(t)$ with one free variable of $FOMLO^-$ such that $\mathcal{M}^-, t \models \phi$ iff $\mathcal{M}_{FO}, t \models \phi(t)$.*
2. *if $\phi \in \Phi^-$, there exist formulae $\phi^{+/\in/-}(t)$ with one free variable such that $\mathcal{M}^-, [ts,te] \models \phi$ iff*

$$\mathcal{M}_{FO}, ts \models \phi^+(ts) \wedge \forall_{>ts}^{<te} t.\ \mathcal{M}_{FO}, t \models \phi^\in(t) \wedge \mathcal{M}_{FO}, te \models \phi^-(te)$$

and, $\mathcal{M}^-, [ts,\infty) \models \phi$ iff $\mathcal{M}_{FO}, ts \models \phi^+(ts) \wedge \forall_{>ts} t.\ \mathcal{M}_{FO}, t \models \phi^\in(t)$

Proof. The proof is straightforward by direct translations. We define the translation $\tau^\bullet$ from formulae of $\Phi^\bullet$ to $FOMLO^-$ formulae as follows:

- $\tau^\bullet(e, x) = e(x)$
- $\tau^\bullet(\neg\phi, x) = \neg\tau^\bullet(\phi, x)$

[5]In what follows, for simplicity, we will refer to atomic predicates of *TPhL* as propositions.

- $\tau^{\bullet}(\phi[\wedge,\vee]\psi,x) = \tau^{\bullet}(\phi,x)[\wedge,\vee]\tau^{\bullet}(\psi,x)$
- $\tau^{\bullet}(\phi \in \psi,x) = \tau^{\bullet}(\phi,x) \wedge (\tau^{-}_{+}(\psi,x) \vee \tau^{-}_{\in}(\psi,x) \vee \tau^{-}_{\in}(\psi,x))$ (We define $\tau^{-}_{+/\in/-}$ below.)

Considering that disjoint intervals can be recreated by their starting points, intermediate and endpoints, we define the $\tau^{-}_{+}, \tau^{-}_{\in}, \tau^{-}_{-}$ translation functions respectively, from formulae of Φ^{-} to *FOMLO*$^{-}$ as follows.

- $\tau^{-}_{\{+,\in,-\}}(s,x) = s_{\{+,\in,-\}}(x)$
- $\tau^{-}_{+}(\phi \rightarrowtail \psi,x) = \tau^{\bullet}(\phi,x) \wedge \forall^{<x} z.\ \left[\tau^{\bullet}(\phi,z) \rightarrow \exists^{\leq t}_{>z} z'.\ \tau^{\bullet}(\psi \wedge \neg\phi, z')\right]$
- $\tau^{-}_{\in}(\phi \rightarrowtail \psi,x) = \exists^{<x} z.\ \left[\tau^{-}_{+}(\phi \rightarrowtail \psi,z) \wedge \forall^{\leq x}_{>z} z'.\ \neg\tau^{\bullet}(\psi \wedge \neg\phi, z')\right]$
- $\tau^{-}_{-}(\phi \rightarrowtail \psi,x) = \tau^{\bullet}(\psi \wedge \neg\phi,x) \wedge \exists^{<x} z.\ \left[\tau^{-}_{+}(\phi \rightarrowtail \psi,z) \wedge \forall^{\leq x}_{>z} z'.\ \neg\tau^{\bullet}(\psi \wedge \neg\phi, z')\right]$
- $\tau^{-}_{+}(\phi \sqcup \psi,x) = \left[\tau^{-}_{+}(\phi,x) \wedge \neg\tau^{-}_{+}(\psi,x) \wedge \neg\tau^{-}_{\in}(\psi,x) \wedge \neg\tau^{-}_{-}(\psi,x)\right] \vee \left[\tau^{-}_{+}(\phi,x) \wedge \tau^{-}_{+}(\psi,x)\right]$
  $\vee \left[\tau^{-}_{+}(\psi,x) \wedge \neg\tau^{-}_{+}(\phi,x) \wedge \neg\tau^{-}_{\in}(\phi,x) \wedge \neg\tau^{-}_{-}(\phi,x)\right]$
- $\tau^{-}_{\in}(\phi \sqcup \psi,x) = \left[\tau^{-}_{+}(\phi,x) \vee \tau^{-}_{\in}(\psi,x)\right] \vee \left[\tau^{-}_{+}(\psi,x) \vee \tau^{-}_{\in}(\phi,x)\right] \vee \left[\tau^{-}_{+}(\phi,x) \wedge \tau^{-}_{-}(\psi,x)\right]$
  $\vee \left[\tau^{-}_{-}(\phi,x) \wedge \tau^{-}_{+}(\psi,x)\right]$
- $\tau^{-}_{-}(\phi \sqcup \psi,x) = \left[\tau^{-}_{-}(\phi,x) \wedge \neg\tau^{-}_{+}(\psi,x) \wedge \neg\tau^{-}_{\in}(\psi,x) \wedge \neg\tau^{-}_{-}(\psi,x)\right] \vee \left[\tau^{-}_{-}(\phi,x) \wedge \tau^{-}_{-}(\psi,x)\right]$
  $\vee \left[\tau^{-}_{-}(\psi,x) \wedge \neg\tau^{-}_{+}(\phi,x) \wedge \neg\tau^{-}_{\in}(\phi,x) \wedge \neg\tau^{-}_{-}(\phi,x)\right]$

The remaining translations are similar to the ones already presented and therefore omitted. It is easy to see that the conditions for an instant to be the starting, intermediate or endpoint of a formula of *TPhL*$^{-}$ is described by *FOMLO*$^{-}$ formulae. Consequently, given a formula ϕ of $\Phi^{\bullet}$, $\mathcal{M}^{-},t \models \phi \leftrightarrow \mathcal{M}_{FO},t \models \tau^{\bullet}(\phi,t)$. Given a formula ϕ of Φ^{-} it holds:

$$\mathcal{M}^{-},[ts,te] \models \phi \text{ iff } \mathcal{M}_{FO},ts \models \tau^{-}_{+}(\phi,ts) \wedge \forall^{<te}_{>ts} t.\ \mathcal{M}_{FO},t \models \tau^{-}_{\in}(\phi,t) \wedge \mathcal{M}_{FO},te \models \tau^{-}_{\in}(\phi,te)$$

Finally, given a formula ϕ of Φ^{-} it holds:

$$\mathcal{M}^{-},[ts,\infty) \models \phi \text{ iff } \mathcal{M}_{FO},ts \models \tau^{-}_{+}(\phi,ts) \wedge \forall_{>ts} t.\ \mathcal{M}_{FO},t \models \tau^{-}_{\in}(\phi,t) \qquad \square$$

From, Theorems 1 and 2 we deduct that:

Theorem 3. *For every formula ϕ of TPhL$_o^{-}$, for all models $\mathcal{M}^{-}$ and their faithful models $\mathcal{M}_{TL}$*[6]*:*

- *if $\phi \in \Phi^{\bullet}$ then there exists a formula ϕ_t of LTL[*`YS`*] such that $\mathcal{M}^{-},t \models \phi$ iff $\mathcal{M}_{TL},t \models \phi_t$.*
- *if $\phi \in \Phi^{-}$ then there exist formulae $\phi_t^{+}, \phi_t^{\in}, \phi_t^{-}$ of LTL[*`YS`*] such that $\mathcal{M}^{-},[ts,te] \models \phi$ iff:*

$$\mathcal{M}_{TL},ts \models \phi_t^{+} \wedge \forall^{<te}_{>ts} t.\ \mathcal{M}_{TL},t \models \phi_t^{\in} \wedge \mathcal{M}_{TL},te \models \phi_t^{-}$$

and, $\mathcal{M}^{-},[ts,\infty) \models \phi$ iff $\mathcal{M}_{TL},ts \models \phi_t^{+} \wedge \forall_{>ts} t.\ \mathcal{M}_{TL},t \models \phi_t^{\in}$

Now we will show that *LTL*[`YS`] is expressible in *TPhL*$_o^{-}$. We define the translation $\tau^{r} : \Phi_t \rightarrow \Phi^{\bullet}$ where Φ_t is the set of formulae of *LTL*[`YS`] and $\Phi^{\bullet}$ is a subset of *TPhL*$_o^{-}$ formulae, as follows:

- $\tau^{r}(p) = p$
- $\tau^{r}(\neg\phi) = \neg\tau^{r}(\phi)$
- $\tau^{r}(\phi\ [\wedge,\vee]\ \psi) = \tau^{r}(\phi)\ [\wedge,\vee]\ \tau^{r}(\psi)$
- $\tau^{r}(\mathtt{Y}\phi) = \big(\tau^{r}(\phi) \vee \neg\tau^{r}(\phi)\big) \in \big(\tau^{r}(\phi) \rightarrowtail \neg\tau^{r}(\phi)\big) \wedge \neg\mathsf{start}\big(\tau^{r}(\phi) \rightarrowtail \neg\tau^{r}(\phi)\big)$
- $\tau^{r}(\phi\ \mathtt{S}\ \psi) = \big(\tau^{r}(\phi) \vee \neg\tau^{r}(\phi)\big) \in \big(\tau^{r}(\psi) \rightarrowtail \neg\tau^{r}(\phi)\big)$

Note that in the case of `Y` and `S`, $\tau^{r}(\phi) \vee \neg\tau^{r}(\phi)$ is true everywhere but is restricted via the $\in$ modality. It is clear that, given a finite set of propositions $\mathcal{P}$ of *LTL*[`YS`], for all models $\mathcal{M}_{TL}$ and their corresponding *faithful TPhL*$_o^{-}$ models (for all $p \in \mathcal{P}$ it holds $V^{TL}(p) = V^{\bullet}(p_e)$ where p_e is an event proposition), and for all *LTL*[`YS`] formulae ϕ it holds $\mathcal{M}_{TL},t \models \phi$ iff $\mathcal{M}^{-},t \models \tau^{r}(\phi)$.

[6]We omit the definition of faithful models of *TPhL* and *LTL* as they are defined in a similar manner to *TPhL* and *FOMLO*.

4.3 *TPhL*$^-$ and *LTL*$[\mathtt{XU},\mathtt{YS}]$

In this section we will show that *TPhL*$^-$, the extension of *TPhL*$_o^-$ (i.e., with filter and $\looparrowright$), has equal expressive power with *LTL*$[\mathtt{XU},\mathtt{YS}]$. Our approach is similar to the previous section, however this time we will translate formulae of *TPhL*$^-$ to *FOMLO* (instead of *FOMLO*$^-$). We will use Kamp's theorem:

Theorem 4 (Kamp [17])**.** *Given any FOMLO formula $\phi(x)$ with one free variable, there is an LTL formula θ, such that $\theta \equiv \phi(x)$ for all models $\mathcal{M}_{FO}$ and $\mathcal{M}_{TL}$.*

Consequently, the only thing that remains to prove that *TPhL*$^-$ is expressible in *LTL*$[\mathtt{XU},\mathtt{YS}]$ is to show that formulae involving the minimal range operator ($\looparrowright$) and filtering (filter) are expressible in *FOMLO*. Similar to the proof of Theorem 2, this is straightforward by extending the translation functions $\tau_+^-, \tau_\in^-, \tau_-^-$ for supporting '$\looparrowright$' and 'filter'. We begin with the translation of formulae that involve the minimal range operator ($\looparrowright$):

$$\tau_+^-(\phi \looparrowright \psi, x) = \tau^\bullet(\phi,x) \wedge \exists_{>x} te.\ \Big[\tau^\bullet(\psi \wedge \neg\phi, te) \wedge \forall_{>x}^{<te} t.\ \big[\neg\tau^\bullet(\phi,x) \wedge \neg\tau^\bullet(\psi \wedge \neg\phi, t)\big]\Big]$$

$$\tau_\in^-(\phi \looparrowright \psi, x) = \exists_{<x} ts. \exists_{>x} te.\ \Big[\tau^\bullet(\phi,ts) \wedge \tau^\bullet(\psi \wedge \neg\phi, te) \wedge \forall_{>ts}^{<te} t.\ \big[\neg\tau^\bullet(\phi,t) \wedge \neg\tau^\bullet(\psi \wedge \neg\phi, t)\big]\Big]$$

$$\tau_-^-(\phi \looparrowright \psi, x) = \exists_{<x} ts. \Big[\tau^\bullet(\phi,ts) \wedge \tau^\bullet(\psi \wedge \neg\phi, x) \wedge \forall_{>ts}^{<x} t.\ \big[\neg\tau^\bullet(\phi,t) \wedge \neg\tau^\bullet(\psi \wedge \neg\phi, t)\big]\Big]$$

Essentially, the translation of $\phi \looparrowright \psi$ is similar to the translation of $\phi \rightarrowtail \psi$, however in this case it is clear that there is a need for future FOMLO formulae. Concerning the translation of formulae involving filtering (filter), first we define formulae $C^{k+}(x_0, \phi, \psi)$ as follows:

$$\begin{aligned} C^{k+}(x_0, \phi, \psi) = & \exists_{>x_0} x_1. \cdots \exists_{>x_{i-1}} x_i. \cdots \exists_{>x_{k-1}} x_k. \nexists_{>x_0}^{<x_1} x_{0,1}. \nexists_{>x_i}^{<x_{i+1}} x_{i,i+1}. \cdots \nexists_{>x_{k-1}}^{<x_k} x_{k-1,k}. \\ & \big[\phi(x_1) \wedge \cdots \wedge \phi(x_{k-1}) \wedge \psi(x_k)\big] \end{aligned}$$

denoting that ϕ is true from x_1 to x_{k-1}, ψ is true at x_k, and all x_i are contiguous and right of x_0. Similar to C^{k+} define the C^{k-} for the left direction from x_0. Here, we will only define the translations for the $\mathsf{filter}_<$ case as the remaining cases can be easily defined in a similar manner.

$$\begin{aligned} \tau_+^-(\phi \ \mathsf{filter}_< \ n, x) = {} & \tau_+^-(\phi,x) \wedge \big[C^{1+}(x, \tau_\in^-(\phi), \tau_-^-(\phi)) \vee \cdots \vee C^{n-1+}(x, \tau_\in^-(\phi), \tau_-^-(\phi))\big] \\ \tau_\in^-(\phi \ \mathsf{filter}_< \ n, x) = {} & \tau_\in^-(\phi,x) \wedge \Big[\big[C^{1-}(x, \tau_\in^-(\phi), \tau_+^-(\phi)) \wedge C^{n-2+}(x, \tau_\in^-(\phi), \tau_-^-(\phi))\big] \\ & \vee \big[C^{2-}(x, \tau_\in^-(\phi), \tau_+^-(\phi)) \wedge C^{n-3+}(x, \tau_\in^-(\phi), \tau_-^-(\phi))\big] \vee \cdots \\ & \vee \big[C^{n-2-}(x, \tau_\in^-(\phi), \tau_+^-(\phi)) \wedge C^{1+}(x, \tau_\in^-(\phi), \tau_-^-(\phi))\big]\Big] \\ \tau_-^-(\phi \ \mathsf{filter}_< \ n, x) = {} & \tau_-^-(\phi,x) \wedge \big[C^{1-}(x, \tau_\in^-(\phi), \tau_+^-(\phi)) \vee \cdots \vee C^{n-1-}(x, \tau_\in^-(\phi), \tau_+^-(\phi))\big] \end{aligned}$$

It can be seen that although a translation of ϕ $\mathsf{filter}_<n$ exists, the size of the translated formula is linear to n. Note that a translation with smaller size might be possible, however for our expressiveness study it is not required to find the optimal translation.

Given all of the above, from Theorem 4, it is clear that the analog of Theorem 3 also holds for *TPhL*$^-$ and *LTL*$[\mathtt{XU},\mathtt{YS}]$. For the opposite direction, it suffices to show that there are τ^r translations from formulae involving the remaining *LTL* modalities, i.e., $\mathtt{X}$ and $\mathtt{U}$, to *TPhL*$^-$ formulae. For convenience we first define $c(\phi) = \phi \rightarrowtail \neg\phi$ where $\phi \in \Phi^\bullet$. Essentially, $c(\phi)$ holds for the maximal intervals $[ts,te]$ or $[ts,\infty)$ for which $\forall_{\geq ts}^{\leq te} t.\mathcal{M},t \models \phi$ or $\forall_{\geq ts} t.\mathcal{M},t \models \phi$ respectively. Therefore we define the corresponding

τ^r translations as follows.

$$\begin{aligned}
\tau^r(\mathtt{X}\phi) &= (\tau^r(\phi) \in c(\tau^r(\phi))) \wedge \neg\mathsf{end}(c(\tau^r(\phi)))) \;\vee\; \mathsf{start}(\neg\tau^r(\phi) \looparrowright \mathsf{end}(c(\neg\tau^r(\phi)))) \\
\tau^r(\phi\ \mathtt{U}\ \psi) &= ((\tau^r(\phi) \vee \neg\tau^r(\phi)) \in (\mathsf{start}(c(\tau^r(\phi)) \sqcap c(\neg\tau^r(\psi))) \looparrowright \\
&\qquad (\mathsf{end}(c(\tau^r(\phi)) \sqcap c(\neg\tau^r(\psi))) \wedge \tau^r(\psi)))) \vee\; \tau^r(\psi)
\end{aligned}$$

Concerning the translation of $\mathtt{X}\phi$, the left part of the disjunction holds true for all instants included in an interval $[ts,te)$ where $c(\tau^r(\phi)))$ is true, while the right part is true at the start of an interval $[t,t+1]$ where $\neg\tau^r(\phi) \looparrowright \mathsf{end}(c(\neg\tau^r(\phi)))$ is true. In the case of $\tau^r(\phi\ \mathtt{U}\ \psi)$, the translation can be divided into two parts: the first part uses the inclusion operator between the tautology $\tau^r(\phi) \vee \neg\tau^r(\phi)$ (true everywhere) and the minimal range formula between (a) the start of a period at which both ϕ and $\neg\psi$ are true for all points (excluding the end) and (b) the end of a period $[ts,te)$ at which both ϕ and $\neg\psi$ are true and ψ holds at te, thus capturing the cases where ϕ is true before ψ becomes true; the second part of the translation is the case of $\tau^r(\psi)$ which captures single instances of ψ. Considering all of the above, we can now say that the *TPhL*$^-$ fragment of *TPhL* has equal expressive power with *LTL*[XU, YS].

4.4 Expressiveness of *TPhL*

Concerning the complete language *TPhL*, a comparison with *LTL* is not possible as the structures on which semantics is based are incomparable, even for atomic entities. This is because dynamic temporal phenomena may hold on non-disjoint intervals which by default require the half plane of a 2-dimensional temporal space for their representation. When compared to *DFO*, it can be easily seen that for all formulae of *TPhL* there are equivalent formulae with two free variables of *DFO*. For example, a dynamic temporal phenomenon proposition p can be represented by a dyadic predicate $p(x,y)$ such that $\mathcal{M},[ts,te] \models p \leftrightarrow \mathcal{M}_{DFO},ts,te \models p(x,y)$. In a similar manner to the previous sections, we can define translations from *TPhL* to *DFO* for the remaining formulae—this time however with two free variables corresponding to starting and ending instants. However, the reverse direction does not hold. This can be shown with the following example. Consider the *DFO* formula $\phi(x,y) = \neg p(x,y)$, since negation is not included for formulae[7] of $\Phi^=$ there is no formula ϕ_t of *TPhL* such that $\mathcal{M},[ts,te] \models \phi_t \leftrightarrow \mathcal{M}_{DFO},ts,te \models \neg p(x,y)$ for all models. Therefore *TPhL* is strictly less expressive than *DFO*.

5 Stream processing

In this section, we formally present the correctness criteria for stream processing with the *TPhL* language. Given a stream, i.e., an arbitrary long sequence of time associated atomic formulae of Φ, the evaluation at a given instant t, of formulae that refer only to the past (ϕ of *TPhL*$_o^-$) is an easy task, as their truth value can be determined for all $t' \leq t$ (see Equation (1)). However, this is not the case for formulae such as $\tau^r(\mathtt{X}p)$ and $\tau^r(p\ \mathtt{U}\ q)$ that refer to the future, as their truth value at an instant t may depend on future information ($> t$). Consequently, in order to guarantee *correctness*, *monotonicity* and *punctuality*—we will define these shortly—, the two valued semantics of *TPhL*, are not sufficient for the evaluation of formulae on constantly evolving streams. In order to treat the issue of evaluations with unknown status—i.e., when all required information is not available at current time—, we follow an approach similar to [6]. We extend the semantics of *TPhL* for stream processing, to utilise three values: true ($\top$), false ($\bot$) and unknown (?). Due to space limitations, we will omit the presentation of the three valued

[7]We chose to omit negation from formulae of $\Phi^=$ as it would affect significantly the performance of our implementation.

semantics[8] in this paper, instead we will focus on formalising the notions of stream, stream processor, and define the properties of correctness, punctuality and monotonicity for stream processors of *TPhL*.

A *stream*, at any instant t can be represented by the finite model $\mathcal{M}_t = \langle T_t, I_t, <, V^{\bullet}, V^{-}, V^{=} \rangle$ where $T_t = \{0, 1, \cdots, t\}$, $I_t = T_t \times T_t \cup \{[ts, \infty) : ts \in T_t\}$, and $V^{\bullet}, V^{-}, V^{=}$ are valuation functions defined in similar manner to Section 2.

A *stream processor*, in symbols $\mathcal{SP}_t$, is defined by the triplet $\langle \Lambda_t^{\bullet}, \Lambda_t^{-}, \Lambda_t^{=} \rangle$ where $t \in T$, $\Lambda_t^{\bullet} : \Phi^{\bullet} \times T_t \rightarrow \{\top, \bot, ?\}$, $\Lambda_t^{-} : \Phi^{-} \times (I_t^c \cup I_T^+) \rightarrow \{\top, \bot, ?\}$, and $\Lambda_t^{=} : \Phi^{=} \times (I_t^c \cup I_t^+) \rightarrow \{\top, \bot, ?\}$, are formulae valuation functions assigning truth values on formulae-instants/intervals pairs and $I_t^c = T_t \times T_t$ and $I_t^+ = \{[ts, t+] : ts, t \in T_t\}$. Intervals of I_t^c, (e.g., [ts,te]) denote that a phenomenon started at ts and ended at te, while intervals of I_t^+, (e.g., $[ts, t+]$) denote that a phenomenon started at ts, and continues to be true/unknown at t but does not end at t. Intervals of I_t^+ are useful for capturing the truth value of valuations that are true but are still ongoing—see for example the semantics of $\phi \rightarrowtail \psi$ for intervals open to ∞. We assume that the input phenomena are ordered and their truth value is never unknown. Now, we define the *correctness*, *punctuality* and *monotonicity* properties for $\mathcal{SP}$.

Correctness. A stream processor has the *correctness* property iff given any stream, for all t and for any $\phi \in \Phi$ evaluation by $\mathcal{SP}_t$ (i.e., via $\Lambda^{\bullet/-/=}$) that is true (false) at an instant t_i or interval i, ϕ is also true (false) (i.e., via the semantics of Section 2) at t_i or i in $\mathcal{M}_t$.

Monotonicity. A stream processor has the *motonocity* property iff given any stream, for all t and for $\phi \in \Phi$ evaluation by $\mathcal{SP}_t$ that is true (false) at an instant t_i or interval i, will also be evaluated to be true (false) at t_i or i by all $\mathcal{SP}_{t'}$ with $t' > t$.

Punctuality. A stream processor has the punctuality property iff given any stream, for any ϕ, and for all instants t_i or intervals i if there exists minimum $t \geq t_i$ such that ϕ is true (false) for all $t' \geq t$ in all $\mathcal{M}_t'$ at t_i or t then $\mathcal{SP}_t$ evaluates ϕ to be true (false) at an instant t_i or interval i.

We say that a stream processor for *TPhL* is *proper* iff it has all the three aforementioned properties. It is easy to see that some formulae, given certain streams, can never be true but always stay unknown. For example, consider the formula $\tau^r(\texttt{G}p)$, where G is the 'globally' *LTL* operator, and a stream where p is true at all instants; at any given point t in time, the stream processor is agnostic to the future, therefore in order to maintain the *monotonicity* property, $\tau^r(\texttt{G}p)$ will be evaluated by $\mathcal{SP}_t$ for all $t' \leq t$ to be unknown.

Phenesthe+, is a *proper* stream processor of *TPhL*. While we will not present a formal proof in this paper, we will briefly discuss its processing and its implementation. Phenesthe+ is a complex event processing engine that given an input stream and a set of temporal phenomena definitions, will produce an output stream of temporal phenomena detections i.e., phenomena associated with a set of instants or intervals at which they are true. Compared to automata based methods, the phenomena are compiled via rewriting into an internal Prolog representation which is then later used for processing. This procedure is linear with respect to the size of the formulae involved. The phenomena definitions can be hierarchical, and their processing, if possible, can happen in parallel. Phenesthe+ detects phenomena by performing temporal queries over tumbling temporal windows of size equal to a user defined step (*ST*) size. A temporal window contains all the new information that has arrived since the last temporal query, as well as information from previous windows that has a possible future use. For example, given the formula $\phi \looparrowright \psi$, if ϕ is true in the current window but ψ is not, then ϕ must be retained. Note that information from previous windows was also retained in the previous version of Phenesthe for valuations that are true but ongoing, or involved dynamic temporal phenomena. All information outside the temporal window that does not have 'future use' is discarded. From a practical perspective it is not viable to keep everything

[8]The complete three valued semantics are available in `https://manospits.github.io/files/Three_valued_semantics.pdf`.

from the past that can contribute to a future detection. Therefore, we allow setting a maximum limit for past information (retaining threshold RT). When this threshold is active Phenesthe+ is no longer *proper* with respect to the full stream, but remains *proper* for the part of the stream that is bounded by RT. Similarly, the *punctuality* property depends on ST, if $ST = 1$, then Phenesthe+ is punctual, however if $ST > 1$, detections will be produced at the latest $ST - 1$ time units after their punctual time.

In terms of complexity, evaluation of formulae in Phenesthe+ happens via single-scan or in the worst case, i.e., when overlapping intervals are involved, polynomial algorithms with respect to the size of the structure (current temporal window). It has to be noted that while *TPhL*$^{-}$ has equal expressive power with LTL, it can accomplish efficient processing of phenomena definitions by utilising intervals to represent set of points. For example an interval $[ts, te]$ produced by the evaluation of the formula $\phi \rightarrowtail \psi$ requires only two points for the representation of all the instants included in $[ts, te]$, therefore contributing significantly to space and processing time economy.

6 Experimental Evaluation

We presented the theoretical basis of *TPhL*. Now, we will evaluate the efficiency of our extended stream processing engine on a reproducible[9] maritime monitoring scenario.

Experimental setup For our experimental evaluation we use a public dataset containing AIS vessel data, transmitted over a period of 6 months, from October 1st, 2015 to March 31st, 2016, in the area of Brest, France [26] along with spatio-temporal events relating vessels with areas (in total $\approx$ 16M input events). We run our experiments on machine with an Intel i7-3770 CPU running Ubuntu 20.04.6 LTS. The set of maritime phenomena we detect as well as the input events are summarised in Table 1. We compare stream processing efficiency when the set of maritime phenomena definitions includes and does not include phenomena marked with ‡, i.e., phenomena that utilise $\looparrowright$ and, or filter or depend on phenomena that utilise them.

Experimental results The results of our evaluation are illustrated in Figure 3. We perform complex event processing with $ST = 3h$, and $RT = \{2, 4, 8, 16\}$ days. Figure 3 (left) shows the average processing time for each experiment. The results show that the addition of ‡ phenomena does not affect processing efficiency significantly, but also that Phenesthe+ is capable of producing detections in less than 2 seconds (multithreaded) when the data retaining threshold is set to 16 days. Note that the performance gain by running the multithreaded version of Phenesthe+ depends on the dependecies between phenomena and the computation of the processing order. For example, *fishing_warning* and *waiting_time* can be processed in parallel while *fishing_warning* and *suspicious_trip* cannot. In [22] we describe the computation of the processing order. With the addition of the new temporal phenomena, we limited the number of phenomena that can be processed in parallel. The results of Figure 3 (left) confirm this. We also perform complex event processing with $ST = 24h$, and set $RT = \infty$ (i.e., keep non-redundant information forever)—recall that our dataset involves a 6 month period. Similar to the previous experiments we compare performance when Phenesthe+ is executed in parallel or serial manner, and with or without temporal phenomena marked with ‡ (see Table 1). Figure 3 (middle) shows the average processing time while Figure 3 (right) shows the average number of input entities plus retained instants/ intervals for each case. The results show, that in terms of processing time, performance is not significantly affected when including ‡ phenomena in both serial and parallel processing even when Phenesthe+ retains all information. In more detail, apart from the input events (on average 90K per temporal query) when we include ‡ phenomena the number of retained instants/intervals increases on average by 20K, therefore bringing the

[9] `https://github.com/manospits/Phenesthe/tree/future`

Table 1: Input and output phenomena description. 'IE', 'UE', 'US' and 'UD' stand for Input/User Event/State/Dynamic temporal phenomenon. Phenomena with † have future dependencies while phenomena with ‡ utilise the new temporal modalities or depend on phenomena that utilise them. The last column lists approximately the number of input or output instants/intervals.

Type	Phenomenon	Description	Number
IE	$ais(V,S,C,H)$	AIS transmitted information (vessel ID, speed, course, heading)	15.8M
	$enters/leaves\{Port, Fishing\}(V,A)$	Vessel enters/leaves port/fishing area.	160K
UE	$stop\_start/end(V)$	Start/end of a stop.	800K
US	$in\_\{port, fishing\}\_area(V,A)$	In fishing/port area.	70K
	$stopped(V)$	Stopped vessel.	300K
	$underway(V)$	Vessel underway.	132K
	$moored(V)$	Moored vessel.	323K
	$^{\dagger\ddagger}fishing\_warning(V,F)$	Warning: Non fishing vessel possibly engaged in fishing.	7K
	$^{\dagger\ddagger}waiting\_time(V,P)$	Port waiting time.	42K
	$^{\dagger\ddagger}long\_waiting\_time$(V, P)	Warning: waiting time longer than a threshold.	28K
	$unusual\_stop(V)$	Warning: vessel performs a stop in an unexpected area.	27K
	$^{\dagger\ddagger}possible\_malfunction(V)$	Warning: Vessel might have a malfunction.	3K
UD	$^{\dagger}trip(V,PA,PB)$	Vessel trip from PA to PB.	39K
	$^{\dagger\ddagger}suspicious\_trip(V,PA,PB)$	Trip from PA to PB contained warnings.	3K
	$^{\dagger}fishing\_trip(V,PA,FA,PB)$	Fishing trip from *PA* to *PB* contained fishing in *FA*.	6K

total number of input+retained entities up to 120K. Even in this setting Phenesthe+, produces detections in approximately 2 and 4 seconds (serial and parallel respectively).

7 Related work & Discussion

There are several very expressive temporal logics. The HS logic [16] is a very powerful logic for representing both instantaneous and durative temporal phenomena. When time is linear and the intervals homogeneous (therefore non overlapping) the HS logic is equally expressive with *LTL* but is exponentially more succinct [9]. In its original version, the HS logic does not make any assumptions on the nature of intervals. In this paper, we showed the $TPhL^{-}$ has equal expressive power to *LTL*, therefore concerning the linear HS variant studied in [9], $TPhL^{-}$ is equally expressive. It is well known, that the chop operator of Venema's CDT logic [27] is inexpressible in the HS logic [20, 10]. *TPhL* supports the chop operator in the form of meets. While a formal expressiveness comparison of *TPhL* with the HS or CDT would be desirable, the omission of negation from $\Phi^{=}$ formulae makes this a challenging task.

Concerning our criteria for proper stream processors of *TPhL*, as mentioned earlier, their concepts are not entirely new. For example "correctness" has a similar notion with "soundness" in run-time verification [5] (i.e., the output should be correct with respect to the specification). Likewise, the "monotonicity" property as we have defined it, in run-time monitoring appears as the irrevocability property respectively for monitors [1]—a monitor that has the "irrevocability" property is unable to revoke the acceptance or the rejection of a trace. Finally, the "punctuality" property can be related to the "tightness" property of monitors [1], under which monitors are restricted to make a choice as soon as there is sufficient information available. In this work, we utilise the similar notions from run-time monitoring and verification for the task of complex event processing.

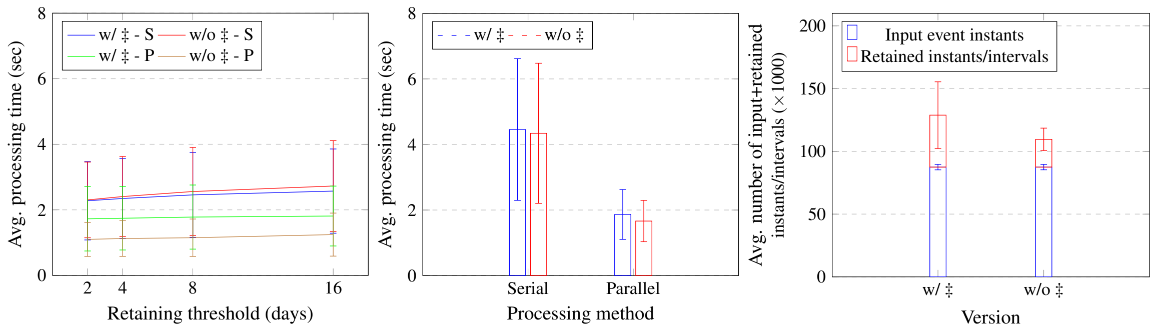


Figure 3: Experimental results. Average processing time per query with $ST = 3h$ and $RT = \{2,4,8,16\}$ days or $ST = 24h$ and $RT = \infty$ (left) and (middle) respectively. Average number of input plus retained instants/ intervals per query when $ST = 24h$ and $RT = \infty$ (right).

From a complex event processing perspective, LARS is a logic based framework for reasoning over streams [7]. While the language of LARS is expressible in *LTL*, the reverse direction does not hold, since LARS does not support 'until'. A well known runtime monitoring system with point-based semantics is LOLA [13]. While LOLA does not allow durative phenomena, we saw in section 4 that formulae that hold on disjoint intervals can be expressed using point based modalities. It is not possible, however, to model formulae that hold on overlapping intervals. Furthermore, in the worst case LOLA requires memory equal to the size of the trace so far, which is not practical for large industrial applications such as maritime monitoring. In Phenesthe+ we allow the user to choose the retaining threshold. A complex event recognition framework is RTEC [4]. RTEC is a logic based formalism whereby events and fluents are expressed with a variant of the Event Calculus [18]. While there isn't a formal study of the expressive power of the language of RTEC, its semantics suggest that it has at most equal expressive power with pure past *LTL*. Bauer et. al. [6], propose three valued semantics for monitoring *LTL* formulae. In our work, we also use three valued semantics, however for a more general case, as our language allows the representation of temporal phenomena that hold on overlapping intervals, which cannot be modeled in *LTL*. Team semantics for LTL [19] or HyperLTL [11] offer a promising direction towards the representation of concurrent temporal phenomena, however they are limited to a finite number of concurrent traces. In *TPhL* a dynamic temporal phenomenon may hold on possibly infinite overlapping intervals.

Closing, in this paper we presented *TPhL* and studied the expressive power of its different fragments. Specifically, we showed that $TPhL_o^-$ has equal expressive power with pure past *LTL* while its extension, $TPhL^-$, has equal expressive power with *LTL*. Concerning the complete logic *TPhL*, we showed that it is strictly less expressive than dyadic first-order logic. Moreover, we defined criteria for *proper* stream processors that use our language, and evaluated Phenesthe+, our stream processing implementation on real maritime data. Our results, show that Phenesthe+ is suitable for the task of maritime monitoring as it produces results in real-time. While the application of our experiment involved the maritime domain, Phenesthe+ is generic, and can be applied in other areas.

Regarding future work, we aim to study the expressive power of a theoretical variant of *TPhL* that includes negation on formulae of $\Phi^=$ in comparison with two-dimensional modal logics. Furthermore, as one of the main motivations for the creation of *TPhL* was facilitating writing temporal formulae, we plan to compare succinctness of $TPhL^-$ formulae with *LTL* formulae. Finally, we aim to apply Phenesthe+ for human activity monitoring in smart homes.

References

[1] Luca Aceto, Antonis Achilleos, Adrian Francalanza, Anna Ingólfsdóttir & Karoliina Lehtinen (2019): *Adventures in Monitorability: From Branching to Linear Time and Back Again*. *Proc. ACM Program. Lang.* 3(POPL), doi:10.1145/3290365.

[2] James F. Allen (1983): *Maintaining knowledge about temporal intervals*. *Communications of the ACM* 26(11), p. 832–843, doi:10.1145/182.358434.

[3] Darko Anicic, Paul Fodor, Sebastian Rudolph, Roland Stühmer, Nenad Stojanovic & Rudi Studer (2010): *A Rule-Based Language for Complex Event Processing and Reasoning*, p. 42–57. *Lecture Notes in Computer Science* 6333, Springer Berlin Heidelberg, doi:10.1007/978-3-642-15918-3_5.

[4] Alexander Artikis, Marek Sergot & Georgios Paliouras (2015): *An Event Calculus for Event Recognition*. *IEEE Transactions on Knowledge and Data Engineering* 27(4), pp. 895–908, doi:10.1109/TKDE.2014.2356476.

[5] Ezio Bartocci & Yliès Falcone, editors (2018): *Lectures on Runtime Verification - Introductory and Advanced Topics*. *Lecture Notes in Computer Science* 10457, Springer, doi:10.1007/978-3-319-75632-5.

[6] Andreas Bauer, Martin Leucker & Christian Schallhart (2011): *Runtime Verification for LTL and TLTL*. *ACM Trans. Softw. Eng. Methodol.* 20(4), doi:10.1145/2000799.2000800.

[7] Harald Beck, Minh Dao-Tran & Thomas Eiter (2018): *LARS: A Logic-based framework for Analytic Reasoning over Streams*. *Artificial Intelligence* 261, pp. 16–70, doi:10.1016/j.artint.2018.04.003.

[8] Michael H Bohlen, Renato Busatto & Christian S. Jensen (1998): *Point-versus interval-based temporal data models*. In: *Proceedings 14th International Conference on Data Engineering*, pp. 192–200, doi:10.1109/ICDE.1998.655777.

[9] Laura Bozzelli, Alberto Molinari, Angelo Montanari, Adriano Peron & Pietro Sala (2018): *Interval vs. Point Temporal Logic Model Checking: An Expressiveness Comparison*. *ACM Trans. Comput. Logic* 20(1), doi:10.1145/3281028.

[10] Davide Bresolin, Dario Della Monica, Valentin Goranko, Angelo Montanari & Guido Sciavicco (2008): *Decidable and Undecidable Fragments of Halpern and Shoham's Interval Temporal Logic: Towards a Complete Classification*. In Iliano Cervesato, Helmut Veith & Andrei Voronkov, editors: *Logic for Programming, Artificial Intelligence, and Reasoning*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 590–604, doi:10.1007/978-3-540-89439-1_41.

[11] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe & César Sánchez (2014): *Temporal Logics for Hyperproperties*. In Martín Abadi & Steve Kremer, editors: *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, *Lecture Notes in Computer Science* 8414, Springer, pp. 265–284, doi:10.1007/978-3-642-54792-8_15.

[12] Gianpaolo Cugola & Alessandro Margara (2010): *TESLA: a formally defined event specification language*. In: *DEBS '10*, ACM Press, p. 50, doi:10.1145/1827418.1827427.

[13] Ben D'Angelo, Sriram Sankaranarayanan, Cesar Sanchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra & Zohar Manna (2005): *LOLA: Runtime Monitoring of Synchronous Systems*. In: *Proceedings of the 12th International Symposium on Temporal Representation and Reasoning*, TIME '05, IEEE Computer Society, USA, p. 166–174, doi:10.1109/TIME.2005.26.

[14] Andreas Dohr, Christiane Engels & Andreas Behrend (2018): *Algebraic Operators for Processing Sets of Temporal Intervals in Relational Databases*. In Natasha Alechina, Kjetil Nørvåg & Wojciech Penczek, editors: *25th International Symposium on Temporal Representation and Reasoning (TIME 2018)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 120, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 11:1–11:16, doi:10.4230/LIPIcs.TIME.2018.11.

[15] Dov Gabbay, Amir Pnueli, Saharon Shelah & Jonathan Stavi (1980): *On the Temporal Analysis of Fairness*. In: *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Program-*

ming Languages, POPL '80, Association for Computing Machinery, New York, NY, USA, p. 163–173, doi:10.1145/567446.567462.

[16] Joseph Y. Halpern & Yoav Shoham (1991): *A propositional modal logic of time intervals*. *Journal of the ACM* 38(4), p. 935–962, doi:10.1145/115234.115351.

[17] Johan Anthony Wilem Kamp (1968): *Tense Logic and the Theory of Linear Order*. Ph.D. thesis, University of California, Los Angeles.

[18] Robert Kowalski & Marek Sergot (1986): *A logic-based calculus of events*. *New Generation Computing* 4(1), pp. 67–95, doi:10.1007/BF03037383.

[19] Andreas Krebs, Arne Meier, Jonni Virtema & Martin Zimmermann (2018): *Team Semantics for the Specification and Verification of Hyperproperties*. In Igor Potapov, Paul G. Spirakis & James Worrell, editors: *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, *LIPIcs* 117, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 10:1–10:16, doi:10.4230/LIPIcs.MFCS.2018.10.

[20] Kamal Lodaya (2000): *Sharpening the Undecidability of Interval Temporal Logic*. In Jifeng He & Masahiko Sato, editors: *Advances in Computing Science - ASIAN 2000, 6th Asian Computing Science Conference, Penang, Malaysia, November 25-27, 2000, Proceedings*, *Lecture Notes in Computer Science* 1961, Springer, pp. 290–298, doi:10.1007/3-540-44464-5_21.

[21] Corto Mascle, Daniel Neider, Maximilian Schwenger, Paulo Tabuada, Alexander Weinert & Martin Zimmermann (2020): *From LTL to RLTL Monitoring: Improved Monitorability through Robust Semantics*. In: *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, HSCC '20, Association for Computing Machinery, New York, NY, USA, pp. 170–204, doi:10.1145/3365365.3382197.

[22] Manolis Pitsikalis, Alexei Lisitsa & Shan Luo (2021): *Representation and Processing of Instantaneous and Durative Temporal Phenomena*. In Emanuele De Angelis & Wim Vanhoof, editors: *Logic-Based Program Synthesis and Transformation - 31st International Symposium, LOPSTR 2021, Tallinn, Estonia, September 7-8, 2021, Proceedings*, *Lecture Notes in Computer Science* 13290, Springer, pp. 135–156, doi:10.1007/978-3-030-98869-2_8.

[23] Manolis Pitsikalis, Alexei Lisitsa, Patrick Totzke & Simon Lee (2022): *Making Sense of Heterogeneous Maritime Data*. In: *2022 23rd IEEE International Conference on Mobile Data Management (MDM)*, pp. 401–406, doi:10.1109/MDM55031.2022.00089.

[24] Amir Pnueli (1977): *The temporal logic of programs*. In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, IEEE, Providence, RI, USA, p. 46–57, doi:10.1109/SFCS.1977.32.

[25] Alexander Rabinovich (2014): *A Proof of Kamp's theorem*. *Logical Methods in Computer Science* Volume 10, Issue 1, doi:10.2168/LMCS-10(1:14)2014.

[26] Cyril Ray, Richard Dréo, Elena Camossi, Anne-Laure Jousselme & Clément Iphar (2019): *Heterogeneous integrated dataset for Maritime Intelligence, surveillance, and reconnaissance*. *Data in Brief* 25, p. 104141, doi:10.1016/j.dib.2019.104141.

[27] Yde Venema (1990): *Expressiveness and completeness of an interval tense logic*. *Notre Dame Journal of Formal Logic* 31(4), doi:10.1305/ndjfl/1093635589.

[28] Peter Øhrstrøm & Per Hasle (2006): *Modern temporal logic: The philosophical background*. In Dov M. Gabbay & John Woods, editors: *Logic and the Modalities in the Twentieth Century*, *Handbook of the History of Logic* 7, North-Holland, pp. 447–498, doi:10.1016/S1874-5857(06)80032-4.

On Two- and Three-valued Semantics for Impure Simplicial Complexes

Hans van Ditmarsch
CNRS, Université de Toulouse, IRIT
Toulouse, France
hans.van-ditmarsch@irit.fr

Roman Kuznets   Rojo Randrianomentsoa
TU Wien
Vienna, Austria
{roman.kuznets+rojo.randrianomentsoa}@tuwien.ac.at[*]

Simplicial complexes are a convenient semantic primitive to reason about processes (agents) communicating with each other in synchronous and asynchronous computation. Impure simplicial complexes distinguish active processes from crashed ones, in other words, agents that are alive from agents that are dead. In order to rule out that dead agents reason about themselves and about other agents, three-valued epistemic semantics have been proposed where, in addition to the usual values true and false, the third value stands for undefined: the knowledge of dead agents is undefined and so are the propositional variables describing their local state. Other semantics for impure complexes are two-valued where a dead agent knows everything. Different choices in designing a semantics produce different three-valued semantics, and also different two-valued semantics. In this work, we categorize the available choices by discounting the bad ones, identifying the equivalent ones, and connecting the non-equivalent ones via a translation. The main result of the paper is identifying the main relevant distinction to be the number of truth values and bridging this difference by means of a novel embedding from three- into two-valued semantics. This translation also enables us to highlight quite fundamental modeling differences underpinning various two- and three-valued approaches in this area of combinatorial topology. In particular, pure complexes can be defined as those invariant under the translation.

1 Introduction

This contribution is on a topic where combinatorial topology and epistemic logic meet, using a categorically motivated duality between simplicial complexes, a structural primitive of a topological nature, and Kripke models, a structural primitive omnipresent in epistemic logic, and modal logics in general. Such simplicial complexes are used to describe synchronous and asynchronous computation, and the focus of our interest is simplicial complexes representing that some processes are alive (active), whereas other processes are dead (have crashed). Such simplicial complexes are called impure. As we will see, their properties can be described in a two-valued epistemic semantics, but also in a three-valued epistemic semantics where the third value means undefined. That represents that formulas cannot be evaluated, as far as the knowledge or other features of a dead agent are concerned. However, there are also ways to represent dead agents in a standard truth-valued (two-valued) semantics: namely, by a dead agent knowing the false proposition (and, therefore, everything — in Kripke semantics it is common to say that such agents are mad, or crazy).

In this contribution we compare two-valued and three-valued epistemic semantics for impure complexes, by way of translations capturing the three-valued meaning in a two-valued way while using the same logical language. We also discuss at length the consequences for truth and validity. These novel translations allow to highlight the respective advantages of such two- and three-valued approaches.

[*]Funded by the Austrian Science Fund (FWF) ByzDEL project (P33600).

In the remainder of this introduction we survey the research area, give detailed informal examples, and summarize the results that we will achieve.

Survey of the literature Combinatorial topology has been used in distributed computing to model concurrency and asynchrony since [5, 16, 27], with higher-dimensional topological properties considered in [24, 23]. The basic structure in combinatorial topology is the *simplicial complex*, a downward closed collection of subsets, called *simplices*, of a set of *vertices*. Geometric manipulations such as subdivision have natural combinatorial counterparts.

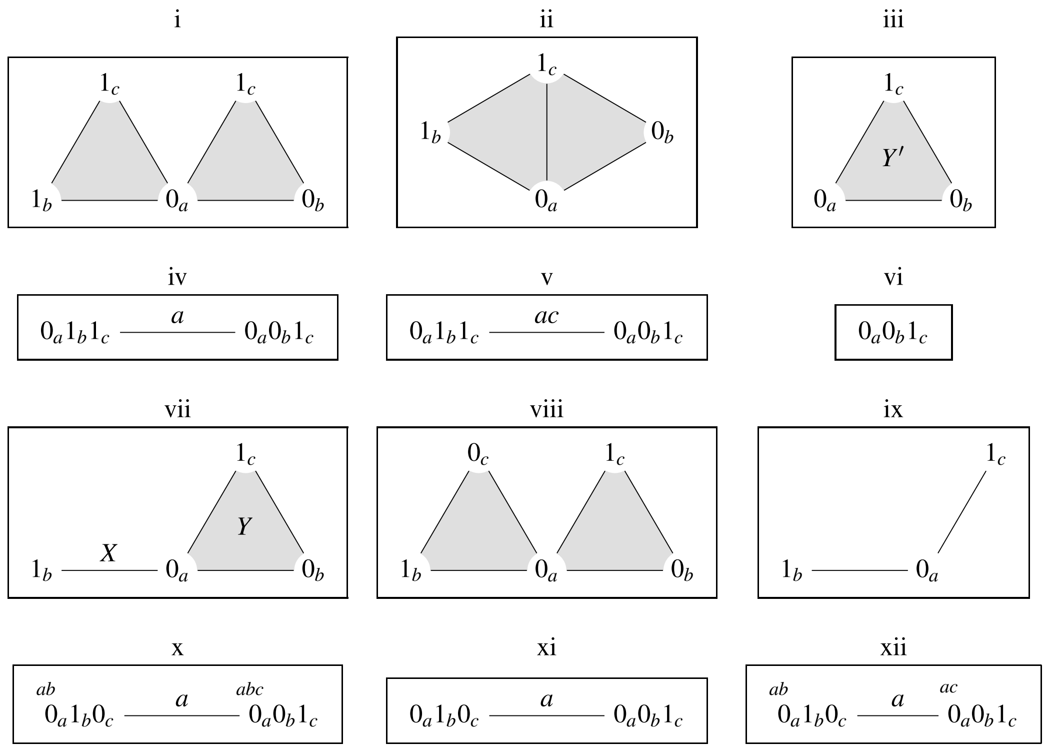
Epistemic logic investigates knowledge and belief, and change of knowledge and belief, in multi-agent systems [25]. Knowledge change was extensively modeled in temporal epistemic logics [1, 22, 30, 13] and more recently in dynamic epistemic logics [3, 11, 28], including synchronous [4] and asynchronous [6, 2] interpretations of dynamics. The basic structure in epistemic semantics is the *Kripke model*, defined by a set of *states* (or *worlds*), a collection of binary *indistinguishability* relations between those states, used to interpret knowledge, and a collection of subsets of states, called *properties*, for where propositional variables are true.

Fairly recently, an epistemic logic interpreted on simplicial complexes was proposed in [26, 20, 8], including exact correspondence between simplicial complexes and Kripke models. Also, in those and other works [29, 33, 9], the action models of [3] are used to model distributed computing tasks and algorithms, with asynchrony treated as in [6]. Action models, which are like Kripke models, also have counterparts that are like simplicial complexes [26, 9].

Even more recently, epistemic semantics for impure complexes have been proposed. In impure complexes some processes have crashed, i.e., are dead. This typically represents synchronous computation (with timeouts), as in asynchronous computation inactive processes can in principle become active again later [23]. Dead agents — and live agents' uncertainty about whether those are dead — need some representation in a modal logical semantics. Choices occurring in the literature are to consider knowledge of dead agents either undefined [12] or trivial [21]. When such knowledge is undefined, it is not allowed to interpret $K_a\varphi$ if a is dead. This results in a three-valued semantics [12] and an accompanying **S5**-like modal logic [31]. When such knowledge is trivial, we mean that $K_a\bot$ is true (agents going mad, or crazy; in epistemic logic, a standard trick coming with an empty accessibility relation [3, 18]), from which we derive that $K_a\varphi$ is true for all formulas φ. This remains a two-valued semantics [21], and the accompanying logic is **KB4**-like. An agent who is dead is not so unlike an agent who is incorrect, as in [17, 7]. The approach of [21] was generalized from individual knowledge to distributed knowledge, and from simplicial complexes to (semi-)simplicial sets [19].

Informal examples Figure 1 displays some simplicial complexes and, for the benefit of the reader more familiar with that representation, corresponding Kripke models. In the depictions of Kripke models we assume reflexivity and symmetry of accessibility relations. The depicted simplicial complexes are for three agents. The vertices of a simplex are required to be labeled with different agents. A maximum size simplex, called facet, therefore, consists of three vertices. This is called dimension 2. These are the triangles in the figure. For two agents we get lines/edges, for four agents we get tetrahedra, etc. A facet corresponds to a state in a Kripke model. A label like 0_a on a vertex represents that it is a vertex for agent a and that agent a's local state has value 0, etc. We can see this as the boolean value of a local proposition where 0 means false and 1 means true. Together these labels determine the valuation in a corresponding Kripke model, for example in states labeled $0_a1_b1_c$ agent a's value is 0, b's is 1, and c's is 1. The single triangle in Fig. 1.iii corresponds to the singleton **S5** model below it, in Fig. 1.vi.

With two triangles, if they only intersect in a, as in Fig. 1.i, it means that agent a cannot distinguish these states, as in Fig. 1.iv, so that a is uncertain about the value of b; whereas if the triangles intersect in a and c, as in Fig. 1.ii, both a and c are uncertain about the value of b, so in corresponding Fig. 1.v the two states are indistinguishable for the two agents a and c.


i
ii
iii
1_c
1_b
0_a
0_b
Y'
iv
v
vi
$0_a1_b1_c$ $\overset{a}{\text{———}}$ $0_a0_b1_c$
$0_a1_b1_c$ $\overset{ac}{\text{———}}$ $0_a0_b1_c$
$0_a0_b1_c$
vii
viii
ix
X
Y
0_c
x
xi
xii
$0_a1_b0_c$ $\overset{a}{\text{———}}$ $0_a0_b1_c$
ab
abc
ac
Figure 1: Examples of pure and impure simplicial complexes and corresponding Kripke models

The current state of the distributed system is represented by a distinguished facet of the simplicial complex, just as we need a distinguished (actual) state in a Kripke model in order to evaluate propositions. For example, in the leftmost triangle of Fig. 1.i, as well as in the leftmost state/world of Fig. 1.iv, a is uncertain whether the value of b is 0 or 1, whereas b knows that its value is 1, and all three agents know that the value of c is 1. However, any face that is not a facet may just as well be taken as the distinguished point of evaluation. For example, in the 0_a vertex of Fig. 1.i it also holds that a is uncertain about the value of c, but b's knowledge is undefined in that vertex. The Kripke model representation in Fig. 1.iv does not allow us such flexibility.

A complex is impure if the maximal faces do not all have the same *maximum* dimension. Let us now consider some impure complexes. Fig. 1.vii consists of two maximal facets, an edge of dimension 1 and a triangle of dimension 2. Therefore, it is impure. Fig. 1.vii represents that agent a is uncertain whether agent c is alive, and also that agent a is uncertain about the value of agent b. The latter is as in Fig. 1.i. However, one might say that a is uncertain whether Fig. 1.vii was "originally" Fig. 1.i or 1.viii, where c's value is 0 on the left. Another impure complex is Fig. 1.ix, wherein a is uncertain whether b is dead or whether c is dead. Although all maximal faces of this complex have the same dimension, this dimension 1 is smaller than 2, the maximum possible for three agents, and that is why this complex is impure. Below these figures we again have shown their corresponding Kripke models, of which we wish to highlight two features. The suffixes with the states indicate which agents are alive. This depiction induces a set of indistinguishability relations: for each agent, the restriction of the domain to the states where the agent is alive is an equivalence relation. This is known as a partial equivalence relation. These are symmetric and transitive relations. The other feature is that, for example, value 0 for c in Fig. 1.x does not correspond to a value for c in edge X in Fig. 1.vii. It is a bogus value. But it does not occur in Fig. 1.vii, which is therefore a more economical representation of the same information. Similarly, for Fig. 1.xii versus Fig. 1.ix. Kripke models only play a minor role in this work. Their relation to complexes is explained in depth in [12].

Finally, we wish to point out a difference, or rather the lack thereof, between Figs. 1.iii and 1.vii. In the only facet Y' of the former, agent a knows that c is alive. In the latter, by contrast, a is uncertain whether c is alive. In the simplicial semantics of [12] this cannot be expressed in the language, which is a feature, not a problem: these facets Y and Y' contain the same information, they make the same formulas true, and the same formulas are defined there. In this work, we will also present a richer semantics that is novel, wherein matters of life and death can be expressed in the language. This is relevant, as these semantics affect translations to two-valued semantics, and thus relate in different ways to works like [21].

Our results We propose two different logical languages and three-valued epistemic semantics for impure simplicial complexes. The primitive modalities are distributed knowledge modalities, of which individual knowledge is a special case. The languages extend each other. The extension consists of propositional variables expressing whether an agent is alive or dead. This cannot be expressed (or defined) in the more restricted language. The extended semantics is novel. For the semantics, it may a priori *seem* to make a difference whether the point of evaluation is a facet (a maximal face) or an arbitrary face. We show that the validities are the same either way, meaning that the logic is insensitive to this choice, and that this is the case for both languages. We then define novel translations relating the three-valued semantics to a two-valued semantics for impure complexes. The crucial aspect is that we can translate 'a formula is defined' into a much larger formula in the same language but interpreted in a two-valued way, where 'a formula is true' and 'a formula is false' more obviously translate to (somewhat shorter) two-valued correspondents. We finally discuss how our results relate to the literature and to further questions.

Overview Section 2 defines the various logical languages and three-valued semantics, for which we then show some properties and give examples. Section 3 defines the two-valued semantics, and shows why similar properties now fail. Section 4 provides translations and proves their adequacy. Examples are also given. Section 5 reviews our results and compares them to the literature.

2 Three-valued epistemic semantics for impure complexes

We consider a finite set A of *agents* (or *processes*) $a,b,\dots$ with $|A| > 1$ and a set $P = \bigcup_{a\in A} P_a$ of *local propositional variables* (or *local atoms*) where P_a are countable and mutually disjoint sets of *local variables for agent a*, denoted $p_a, q_a, p'_a, q'_a, \dots$ We also view the agent's name a as a *global propositional variable* (or *global atom*) stating that agent a is alive.

We successively define the logical languages, simplicial complexes, and related structural notions, and then give the semantics.

Definition 1 (Languages)**.** *Language $\mathscr{L}_D^{\mathsf{gloc}}$ for distributed knowledge with glocal*[1] *atoms* is defined by

$$\varphi ::= a \mid p_a \mid \neg\varphi \mid (\varphi \land \varphi) \mid \widehat{D}_B\varphi \tag{1}$$

[1]Glocal is a portmanteau word formed from global+local.

where $a \in A$, $B \subseteq A$, and $p_a \in P_a$. Apart from other propositional connectives expressed via the standard notational abbreviations, we use $D_B\varphi := \neg\widehat{D}_B\neg\varphi$, $\widehat{K}_a\varphi := \widehat{D}_{\{a\}}\varphi$, and $K_a\varphi := \neg\widehat{K}_a\neg\varphi$. The last two abbreviations mean that individual knowledge K_a of agent a is naturally expressed in this language as distributed knowledge $D_{\{a\}}$ of the singleton group $\{a\}$. *Language $\mathscr{L}_D^{\mathsf{loc}}$ for distributed knowledge with local atoms* is obtained from grammar (1) by dropping global atoms a. *Language $\mathscr{L}_K^{\mathsf{gloc}}$ for individual knowledge with glocal atoms* is the language where sets $B \subseteq A$ are restricted to singleton sets and K_a and $\widehat{K}_a$ are used instead of $D_{\{a\}}$ and $\widehat{D}_{\{a\}}$ respectively. *Language $\mathscr{L}_K^{\mathsf{loc}}$ for individual knowledge with local atoms* is both restricted to individual knowledge and without global atoms.

For $D_B\varphi$ we read 'group B of agents have distributed knowledge of φ,' and for $K_a\varphi$ we read 'agent a knows φ.'

Next, we define our structural primitive, the simplicial model. Other than in the introduction, Kripke models play no part in this work and will not be defined.

Definition 2 (Simplicial model)**.** A *simplicial model* $\mathscr{C}$ is a triple (C,χ,ℓ) where C is a *simplicial complex*, χ is a *chromatic map*, and ℓ is a *valuation*. Here:

- A *(simplicial) complex* $C \neq \varnothing$ is a collection of *simplices* that are non-empty finite subsets of a given set $\mathscr{V}$ of vertices such that C is downward closed (i.e., $X \in C$ and $\varnothing \neq Y \subseteq X$ imply $Y \in C$). Simplices represent partial global states of a distributed system. It is required that every vertex form a simplex by itself, i.e., $\big\{\{v\} \mid v \in \mathscr{V}\big\} \subseteq C$.
- Vertices represent local states of agents, with a *chromatic map* $\chi\colon \mathscr{V} \to A$ assigning each vertex to one of the agents in such a way that each agent has at most one vertex per simplex, i.e., $\chi(v) = \chi(u)$ for some $v,u \in X \in C$ implies that $v = u$. For $X \in C$, we define $\chi(X) := \{\chi(v) \mid v \in X\}$ to be the set of agents in simplex X.
- A *valuation* $\ell\colon \mathscr{V} \to 2^P$ assigns to each vertex which local variables of the vertex's owner are true in it, i.e., $\ell(v) \subseteq P_a$ whenever $\chi(v) = a$. Variables from $P_a \setminus \ell(v)$ are false in vertex v, whereas variables from $P \setminus P_a$ do not belong to agent a and cannot be evaluated in a's vertex v. The set of variables that are true in simplex $X \in C$ is given by $\ell(X) := \bigcup_{v\in X} \ell(v)$.

If $Y \subseteq X$ for $X,Y \in C$, we say that Y is a *face* of X. Since each simplex is a face of itself, we use 'simplex' and 'face' interchangeably. A face X is a *facet* iff it is a maximal simplex in C, i.e., $Y \in C$ and $Y \supseteq X$ imply $Y = X$. Facets represent global states of the distributed system, and their set is denoted $\mathscr{F}(C)$. The *dimension of simplex* X is $|X| - 1$, e.g., vertices are of dimension 0, edges are of dimension 1, etc. The *dimension of a simplicial complex (model)* is the largest dimension of its facets. A simplicial complex (model) is *pure* iff all facets have dimension n where $|A| = n+1$, i.e., contain vertices for all agents. Otherwise it is *impure*. A *pointed simplicial model* is a pair $(\mathscr{C},X)$ where $X \in C$.

Having defined the logical language and the structures, we now present the three-valued semantics. We distinguish *face-semantics* that are interpreted on arbitrary faces of simplicial models from *facet-semantics* that are only interpreted on facets (maximal faces). We will later prove that the three-valued face- and facet-semantics determine the same validities, so that the difference does not matter. However, subsequently we show that for the two-valued semantics the difference matters a great deal.

Definition 3 (Three-valued definability and satisfaction relations)**.** The *definability relation* $\bowtie$ and *satisfaction relation* $\Vdash$ are defined by induction on $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$. Let $\mathscr{C} = (C,\chi,\ell)$ be a simplicial model and

$X \in C$ a face.

$$\begin{array}{lll}
\mathscr{C},X \bowtie a & \text{iff} & X \in \mathscr{F}(C); \\
\mathscr{C},X \bowtie p_a & \text{iff} & a \in \chi(X); \\
\mathscr{C},X \bowtie \neg\varphi & \text{iff} & \mathscr{C},X \bowtie \varphi; \\
\mathscr{C},X \bowtie \varphi \wedge \psi & \text{iff} & \mathscr{C},X \bowtie \varphi \text{ and } \mathscr{C},X \bowtie \psi; \\
\mathscr{C},X \bowtie \widehat{D}_B\varphi & \text{iff} & \mathscr{C},Y \bowtie \varphi \text{ for some } Y \in C \text{ with } B \subseteq \chi(X \cap Y).
\end{array}$$

$$\begin{array}{lll}
\mathscr{C},X \Vdash a & \text{iff} & \mathscr{C},X \bowtie a \text{ and } a \in \chi(X); \\
\mathscr{C},X \Vdash p_a & \text{iff} & p_a \in \ell(X); \\
\mathscr{C},X \Vdash \neg\varphi & \text{iff} & \mathscr{C},X \bowtie \neg\varphi \text{ and } \mathscr{C},X \nVdash \varphi; \\
\mathscr{C},X \Vdash \varphi \wedge \psi & \text{iff} & \mathscr{C},X \Vdash \varphi \text{ and } \mathscr{C},X \Vdash \psi; \\
\mathscr{C},X \Vdash \widehat{D}_B\varphi & \text{iff} & \mathscr{C},Y \Vdash \varphi \text{ for some } Y \in C \text{ with } B \subseteq \chi(X \cap Y).
\end{array}$$

A formula $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$ is *valid* (and we write $\Vdash \varphi$) iff for any simplicial model $\mathscr{C} = (C,\chi,\ell)$ and face $X \in C$, we have that $\mathscr{C},X \bowtie \varphi$ implies $\mathscr{C},X \Vdash \varphi$.

The face-semantics for the other three languages can be derived from Definition 3 by restricting the formulas interpreted correspondingly (see Definition 1).[2]

The semantics for the dual, box-like modality D_B can be derived from the above and is slightly more complex and less intuitive in this three-valued setting. This is why we use the diamond-like $\widehat{D}_B$ as a primitive. Note that, as any φ is definable iff $\neg\varphi$ is definable, $\mathscr{C},Y \bowtie D_B\varphi$ iff $\mathscr{C},Y \bowtie \widehat{D}_B\varphi$.

$$\mathscr{C},X \Vdash D_B\varphi \quad \text{iff} \quad \mathscr{C},Y \bowtie \widehat{D}_B\varphi \text{ and } (\mathscr{C},Y \bowtie \varphi \Rightarrow \mathscr{C},Y \Vdash \varphi) \text{ for all } Y \in C \text{ with } B \subseteq \chi(X \cap Y).$$

The facet-semantics can be derived from Definition 3 by evaluating formulas on facets only. We denote such three-valued facet-semantics by writing $\bowtie^{\mathscr{F}}$ and $\Vdash^{\mathscr{F}}$ instead of $\bowtie$ and $\Vdash$. All clauses are the same, except for $X \in \mathscr{F}(C)$,

$$\mathscr{C},X \bowtie^{\mathscr{F}} \widehat{D}_B\varphi \quad \text{iff} \quad \mathscr{C},Y \bowtie^{\mathscr{F}} \varphi \text{ for some } Y \in \mathscr{F}(C) \text{ with } B \subseteq \chi(X \cap Y); \tag{2}$$

$$\mathscr{C},X \Vdash^{\mathscr{F}} \widehat{D}_B\varphi \quad \text{iff} \quad \mathscr{C},Y \Vdash^{\mathscr{F}} \varphi \text{ for some } Y \in \mathscr{F}(C) \text{ with } B \subseteq \chi(X \cap Y). \tag{3}$$

Validity in the facet-semantics is defined as follows: formula $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$ is *valid*, written $\Vdash^{\mathscr{F}} \varphi$, iff for any simplicial model $\mathscr{C} = (C,\chi,\ell)$ and facet $X \in \mathscr{F}(C)$, we have $\mathscr{C},X \bowtie^{\mathscr{F}} \varphi$ implies $\mathscr{C},X \Vdash^{\mathscr{F}} \varphi$.

Various results for language $\mathscr{L}_K^{\mathsf{loc}}$ reported in [12] directly generalize to $\mathscr{L}_D^{\mathsf{loc}}$, $\mathscr{L}_D^{\mathsf{gloc}}$, and $\mathscr{L}_K^{\mathsf{gloc}}$. We, therefore, give those without proof: the semantics of $\to$, $\leftrightarrow$, and $\vee$ are not standardly boolean but require that both arguments be defined; truth implies definability; and duality is valid:

Lemma 4 ([12]). *1.* $\mathscr{C},X \Vdash \varphi \vee \psi$ *iff* $\mathscr{C},X \bowtie \varphi$ *and* $\mathscr{C},X \bowtie \psi$ *and* $(\mathscr{C},X \Vdash \varphi$ *or* $\mathscr{C},X \Vdash \psi)$*;*

2. $\mathscr{C},X \Vdash \varphi \to \psi$ *iff* $\mathscr{C},X \bowtie \varphi$ *and* $\mathscr{C},X \bowtie \psi$ *and* $(\mathscr{C},X \Vdash \varphi \Rightarrow \mathscr{C},X \Vdash \psi)$*;*

3. $\mathscr{C},X \Vdash \varphi \leftrightarrow \psi$ *iff* $\mathscr{C},X \bowtie \varphi$ *and* $\mathscr{C},X \bowtie \psi$ *and* $(\mathscr{C},X \Vdash \varphi$ *iff* $\mathscr{C},X \Vdash \psi)$*;*

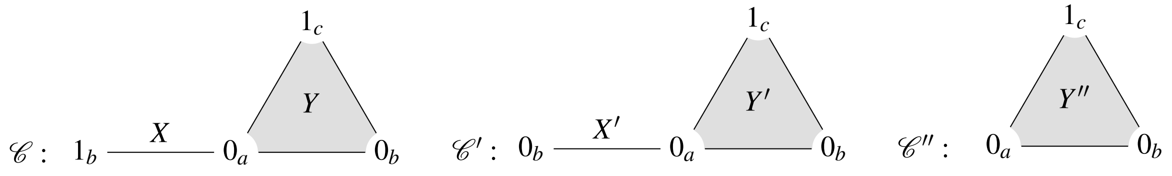
4. $\mathscr{C},X \Vdash \varphi \quad \Rightarrow \quad \mathscr{C},X \bowtie \varphi$*;*

5. $\Vdash D_B\varphi \leftrightarrow \neg\widehat{D}_B\neg\varphi$.

The same properties hold for $\bowtie^{\mathscr{F}}$ *and* $\Vdash^{\mathscr{F}}$.

[2] Alternatively to declaring global atoms a definable in facets only, we could have made a definable in facets and in any face where a is alive. This would not have affected any results we report. We, therefore, chose the conceptually "cleaner" option of global variables defined only in global states.

Example 5. Consider the following simplicial models $\mathscr{C}$, $\mathscr{C}'$, and $\mathscr{C}''$ for three agents a, b, and c with their values represented by local variables p_a, p_b, and p_c respectively.



- Atoms and knowledge of dead agents: Illustrating the novel aspects of the semantics, we have that $\mathscr{C},X \not\bowtie p_c$ since $c \notin \chi(X)$. Consequently, $\mathscr{C},X \not\bowtie \neg p_c$, $\mathscr{C},X \not\Vvdash p_c$, and $\mathscr{C},X \not\Vvdash \neg p_c$. Besides, again because $c \notin \chi(X)$, we have that $\mathscr{C},X \not\bowtie \widehat{K}_c \neg p_a$. Therefore, $\mathscr{C},X \not\bowtie \neg\widehat{K}_c \neg p_a$, $\mathscr{C},X \not\Vvdash \widehat{K}_c \neg p_a$, and $\mathscr{C},X \not\Vvdash \neg\widehat{K}_c \neg p_a$.

- Knowledge of a live agent concerning dead agents: Although $\mathscr{C},X \not\bowtie p_c$, after all $\mathscr{C},X \Vvdash \widehat{K}_a p_c$ because $a \in \chi(X \cap Y)$ and $\mathscr{C},Y \Vvdash p_c$: agent a considers it possible that agent c is alive. More surprisingly, also $\mathscr{C},X \Vvdash K_a p_c$ because given the two facets X and Y that agent a considers possible (and all of their faces), as far as a knows, p_c is true. This knowledge is defeasible because a may learn that the actual facet is X and not Y, which she also considers possible.

  We further have that $\mathscr{C},X \not\Vvdash K_a p_c \to p_c$ because $\mathscr{C},X \not\bowtie p_c$ implies $\mathscr{C},X \not\bowtie K_a p_c \to p_c$. However, $\Vvdash K_a p_c \to p_c$ because whenever $K_a p_c \to p_c$ is defined, the truth of $K_a p_c$ implies the truth of p_c.

- Individual and distributed knowledge: As expected, we have $\mathscr{C},X \Vvdash p_b \wedge \neg p_a$, where the conjunct $\mathscr{C},X \Vvdash \neg p_a$ is justified by $\mathscr{C},X \bowtie p_a$ and $\mathscr{C},X \not\Vvdash p_a$. We also have $\mathscr{C},Y \Vvdash \widehat{K}_a p_b$ because $a \in \chi(X \cap Y)$ and $\mathscr{C},X \Vvdash p_b$. At the same time, $\mathscr{C},Y \Vvdash D_{\{a,b\}} \neg p_b$ because all faces Z such that $\{a,b\} \subseteq \chi(Z \cap Y)$ share the 0_a–0_b edge with Y and, hence, have $\mathscr{C},Z \Vvdash \neg p_b$.

  Considering the global atom c, we have that $\mathscr{C},X \not\Vvdash K_a \neg c$ because $a \in \chi(X \cap Y)$ and $\mathscr{C},Y \not\Vvdash \neg c$. However, $\mathscr{C},X \Vvdash D_{\{a,b\}} \neg c$. In fact, $\mathscr{C},X \Vvdash \varphi$ iff $\mathscr{C},X \Vvdash D_{\{a,b\}} \varphi$: a and b have distributed omniscience on edge X.

- Local and glocal languages: All formulas of language $\mathscr{L}_D^{\mathsf{loc}}$ are undefined/true/false in $(\mathscr{C}',Y')$ iff they are undefined/true/false in $(\mathscr{C}'',Y'')$ [12]. In other words, it is impossible to express in $\mathscr{L}_D^{\mathsf{loc}}$ that a considers it possible that c is dead. By contrast, in language $\mathscr{L}_D^{\mathsf{gloc}}$ we have $\mathscr{C}',Y' \Vvdash \widehat{K}_a \neg c$ whereas $\mathscr{C}'',Y'' \not\Vvdash \widehat{K}_a \neg c$. Hence, global atoms make the language more expressive.

Upwards and downwards monotonicity for the face-semantics requires certain care and is, therefore, given in some detail for the extended languages.

Lemma 6 (Monotonicity)**.** *For a simplicial model $\mathscr{C} = (C,\chi,\ell)$, faces $X,Y \in C$ with $X \subseteq Y$, and formula $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$,*

1. *$\mathscr{C},X \bowtie \varphi$ implies $\mathscr{C},Y \bowtie \varphi$;* *(upwards monotonicity of definability)*
2. *$\mathscr{C},X \Vvdash \varphi$ implies $\mathscr{C},Y \Vvdash \varphi$;* *(upwards monotonicity of truth)*
3. *$\mathscr{C},Y \Vvdash \varphi$ and $\mathscr{C},X \bowtie \varphi$ imply $\mathscr{C},X \Vvdash \varphi$.* *(downwards monotonicity of truth modulo definability)*

Proof. The proof is by induction on $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$. All cases are as in [12] except the new cases for global atoms and (dual) distributed knowledge that are shown below:

Case a Whether $\mathscr{C},X \bowtie a$ or $\mathscr{C},X \Vvdash a$ is assumed, X must be a facet, so $Y = X$ is the same facet. Consequently, $\mathscr{C},X \bowtie \varphi$ implies $\mathscr{C},Y \bowtie \varphi$, $\mathscr{C},X \Vvdash \varphi$ implies $\mathscr{C},Y \Vvdash \varphi$, and $\mathscr{C},Y \Vvdash \varphi$ and $\mathscr{C},X \bowtie \varphi$ imply $\mathscr{C},X \Vvdash \varphi$.

Case $\widehat{D}_B\varphi$

1. Assume that $\mathscr{C},X \bowtie \widehat{D}_B\varphi$. Then $\mathscr{C},Z \bowtie \varphi$ for some $Z \in C$ with $B \subseteq \chi(X \cap Z)$. Since $X \subseteq Y$, we have $X \cap Z \subseteq Y \cap Z$ and, therefore, $B \subseteq \chi(Y \cap Z)$. It follows that $\mathscr{C},Y \bowtie \widehat{D}_B\varphi$.
2. Assume that $\mathscr{C},X \Vdash \widehat{D}_B\varphi$. Then $\mathscr{C},Z \Vdash \varphi$ for some $Z \in C$ with $B \subseteq \chi(X \cap Z)$. Since $X \subseteq Y$, we have $X \cap Z \subseteq Y \cap Z$ and, therefore, $B \subseteq \chi(Y \cap Z)$. It follows that $\mathscr{C},Y \Vdash \widehat{D}_B\varphi$.
3. Assume that $\mathscr{C},Y \Vdash \widehat{D}_B\varphi$ and $\mathscr{C},X \bowtie \widehat{D}_B\varphi$. Then $\mathscr{C},Z \Vdash \varphi$ for some $Z \in C$ with $B \subseteq \chi(Y \cap Z)$. Additionally, $B \subseteq \chi(X)$ because $\mathscr{C},X \bowtie \widehat{D}_B\varphi$. Due to χ being chromatic, any vertex v with $\chi(v) \in B$ that belongs to Y must belong to X. Hence, $B \subseteq \chi(X \cap Z)$ and, therefore, $\mathscr{C},X \Vdash \widehat{D}_B\varphi$. □

Remark 7. Note that this lemma becomes trivial for $\bowtie^{\mathscr{F}}$ and $\Vdash^{\mathscr{F}}$ since $X \subseteq Y$ for facets implies $X = Y$.

An axiomatization of validities in $\mathscr{L}_K^{\mathsf{loc}}$ is reported in [31], but none so far for $\mathscr{L}_K^{\mathsf{gloc}}$, which is novel in this contribution. Axiomatizations of the distributed knowledge versions $\mathscr{L}_D^{\mathsf{loc}}$ and $\mathscr{L}_D^{\mathsf{gloc}}$ should extend the axiomatizations of distributed knowledge [15, 32, 10] and will likely be related to [19].

Comparing the three-valued face- to facet-semantics We continue by showing that the set of validities is the same for the three-valued face- and facet-semantics. This (novel) result did not initially seem obvious to us, and it plays an important role when embedding the three-valued into two-valued semantics, as the face-semantics is infelicitous in the latter. We show the results for $\mathscr{L}_D^{\mathsf{gloc}}$. The results for sublanguages $\mathscr{L}_D^{\mathsf{loc}}$, $\mathscr{L}_K^{\mathsf{gloc}}$, and $\mathscr{L}_K^{\mathsf{loc}}$ follow directly.

Lemma 8. $\mathscr{C},X \bowtie^{\mathscr{F}} \varphi \quad\Leftrightarrow\quad \mathscr{C},X \bowtie \varphi$ *for any* $\mathscr{C} = (C,\chi,\ell)$, $X \in \mathscr{F}(C)$, *and* $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$.

Proof. The proof is by induction on $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$. The two semantics coincide for variables and propositional connectives (in particular, global variables are defined on all facets and only on them according to both). The only non-trivial case to consider is $\widehat{D}_B\varphi$.

$\Leftarrow$ Assume that $\mathscr{C},X \bowtie \widehat{D}_B\varphi$. Then $\mathscr{C},Y \bowtie \varphi$ for some face $Y \in C$ with $B \subseteq \chi(X \cap Y)$. But $Y \subseteq Z$ for some facet $Z \in \mathscr{F}(C)$ and we have $X \cap Y \subseteq X \cap Z$. Hence, $B \subseteq \chi(X \cap Z)$ and $\mathscr{C},Z \bowtie \varphi$ by Lemma 6.1. It follows from IH that $\mathscr{C},Z \bowtie^{\mathscr{F}} \varphi$. Thus, $\mathscr{C},X \bowtie^{\mathscr{F}} \widehat{D}_B\varphi$.

$\Rightarrow$ Assume $\mathscr{C},X \bowtie^{\mathscr{F}} \widehat{D}_B\varphi$. This direction is even simpler as here facet $Y \in \mathscr{F}(C)$ is itself a face. □

Lemma 9. $\mathscr{C},X \Vdash^{\mathscr{F}} \varphi \quad\Leftrightarrow\quad \mathscr{C},X \Vdash \varphi$ *for any* $\mathscr{C} = (C,\chi,\ell)$, $X \in \mathscr{F}(C)$, *and* $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$.

Proof. The proof is by induction on $\varphi \in \mathscr{L}_D^{\mathsf{loc}}$. Again the two semantics coincide except for $\widehat{D}_B\varphi$.

$\Leftarrow$ Assume that $\mathscr{C},X \Vdash \widehat{D}_B\varphi$. Then $\mathscr{C},Y \Vdash \varphi$ for some face $Y \in C$ with $B \subseteq \chi(X \cap Y)$. But $Y \subseteq Z$ for some facet $Z \in \mathscr{F}(C)$ and we have $X \cap Y \subseteq X \cap Z$. Hence, $B \subseteq \chi(X \cap Z)$ and $\mathscr{C},Z \Vdash \varphi$ by Lemma 6.2. We can conclude that $\mathscr{C},X \Vdash^{\mathscr{F}} \widehat{D}_B\varphi$.

$\Rightarrow$ Again this case is similar but simpler as every facet $Y \in \mathscr{F}(C)$ is a face. □

Theorem 10. $\Vdash^{\mathscr{F}} \varphi \quad\Leftrightarrow\quad \Vdash \varphi$ *for any* $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$.

Proof. $\Rightarrow$ Assume that $\Vdash^{\mathscr{F}} \varphi$ and consider an arbitrary simplicial model $\mathscr{C} = (C,\chi,\ell)$ and an arbitrary face $X \in C$ with $\mathscr{C},X \bowtie \varphi$. Then, $X \subseteq Y$ for some facet $Y \in \mathscr{F}(C)$. We have $\mathscr{C},Y \bowtie \varphi$ by Lemma 6.1 and $\mathscr{C},Y \bowtie^{\mathscr{F}} \varphi$ by Lemma 8. Since $\Vdash^{\mathscr{F}} \varphi$, we have $\mathscr{C},Y \Vdash^{\mathscr{F}} \varphi$. It follows from Lemma 9 that $\mathscr{C},Y \Vdash \varphi$. Hence, $\mathscr{C},X \Vdash \varphi$ by Lemma 6.3. Thus, $\Vdash \varphi$.

$\Leftarrow$ Assume that $\Vdash \varphi$. Then $\mathscr{C},X \bowtie \varphi$ implies $\mathscr{C},X \Vdash \varphi$ for any face $X \in C$ of any simplicial model $\mathscr{C} = (C,\chi,\ell)$. In particular, this is the case for all facets of any simplicial complex $\mathscr{C}$. We can conclude from Lemmas 8 and 9 that $\Vdash^{\mathscr{F}} \varphi$. □

Remark 11. Given that the restriction to facets does not affect the logic of the three-valued semantics, it is worth noting that boolean constants $\top$ and $\bot$ are expressible in $\mathscr{L}_K^{\mathsf{gloc}}$ and $\mathscr{L}_D^{\mathsf{gloc}}$ in the facet- but not in the face-semantics. Indeed, there is no formula defined in all faces of all simplicial models. Hence, no formula can be always true as $\top$ or always false as $\bot$. By contrast, $a \vee \neg a$ can serve as $\top$ and $a \wedge \neg a$ as $\bot$ for the facet-semantics. Languages $\mathscr{L}_K^{\mathsf{loc}}$ and $\mathscr{L}_D^{\mathsf{loc}}$, on the other hand, cannot express boolean constants in any three-valued semantics.

We have shown that three-valued semantics is robust with respect to definability in that the truth value of a formula does not depend on which of the agents are crashed as long as the formula is defined. In addition, the monotonicity of definability makes it possible to restrict attention to facets only, in line with the understanding that only they represent actual global states of the distributed system. The stability of the three-valued semantics modulo the choice of a partial global state or the restriction to global states only is, in our view, a strong argument in its favor.

3 Two-valued epistemic semantics for impure complexes

We now present a two-valued semantics for impure complexes. It is inspired by that in [21] (to which we will compare it in the final Sect. 5), but in this work its role is rather that of a technical tool to enable us to embed three-valued semantics, and to explain why choices essential in the three-valued setting are infelicitous or non-existent in the two-valued one.

Without the third truth value "undefined," definability plays no role: every formula is defined in every face. We, therefore, need to define only the satisfaction relation. The languages are the same. Further simplifying the two-valued setting, we will show that the global propositional variables of $\mathscr{L}_D^{\mathsf{gloc}}$ are expressible in the restricted language $\mathscr{L}_D^{\mathsf{loc}}$, which therefore suffices.

To distinguish the two-valued from three-valued semantics we write $\Vdash$ for the former to contrast it with $\Vvdash$ that we used for the latter. An astute reader might notice that we use the notation $\Vdash$ with two vertical lines for the two-valued semantics and $\Vvdash$ with three vertical lines for the three-valued one to make the distinction obvious.

Definition 12 (Two-valued facet satisfaction relation)**.** We define the satisfaction relation $\Vdash^{\mathscr{F}}$ by induction on $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$. Let $\mathscr{C} = (C, \chi, \ell)$ and $X \in \mathscr{F}(C)$.

$$\begin{array}{lll}
\mathscr{C}, X \Vdash^{\mathscr{F}} a & \text{iff} & a \in \chi(X) \\
\mathscr{C}, X \Vdash^{\mathscr{F}} p_a & \text{iff} & p_a \in \ell(X) \\
\mathscr{C}, X \Vdash^{\mathscr{F}} \varphi \wedge \psi & \text{iff} & \mathscr{C}, X \Vdash^{\mathscr{F}} \varphi \text{ and } \mathscr{C}, X \Vdash^{\mathscr{F}} \psi \\
\mathscr{C}, X \Vdash^{\mathscr{F}} \neg \varphi & \text{iff} & \mathscr{C}, X \nVdash^{\mathscr{F}} \varphi \\
\mathscr{C}, X \Vdash^{\mathscr{F}} \widehat{D}_B \varphi & \text{iff} & \mathscr{C}, Y \Vdash^{\mathscr{F}} \varphi \text{ for some } \; Y \in \mathscr{F}(C) \text{ with } B \subseteq \chi(X \cap Y)
\end{array}$$

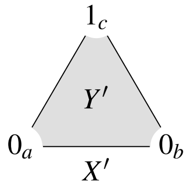
As the superscript $\mathscr{F}$ suggests, this is a semantics for facets. The reason we give it as the primary in the two-valued case rather than considering alongside the semantics $\Vdash$ for arbitrary faces, as we did for three values, is that the latter is infelicitous, as we show in Prop. 14.

Remark 13. For all four languages, $\top := p_a \vee \neg p_a$ and $\bot := p_a \wedge \neg p_a$ can serve as boolean constants in the two-valued semantics.

In contrast to the three-valued semantics $\Vvdash^{\mathscr{F}}$, the semantics of, for example, implication is now the standard boolean semantics so that $\mathscr{C}, X \Vdash^{\mathscr{F}} \varphi \rightarrow \psi$ iff $\mathscr{C}, X \Vdash^{\mathscr{F}} \varphi$ implies $\mathscr{C}, X \Vdash^{\mathscr{F}} \psi$. It is simply the version without definability requirements. Similarly, for other propositional connectives and for the dual distributed knowledge modality:

$$\mathscr{C}, X \Vdash^{\mathscr{F}} D_B \varphi \quad \text{iff} \quad \mathscr{C}, Y \Vdash^{\mathscr{F}} \varphi \text{ for all } Y \in \mathscr{F}(C) \text{ with } B \subseteq \chi(X \cap Y).$$

One of the important resulting differences is that, in contrast to the three-valued semantics $\Vvdash$ (see Lemma 6), the two-valued face-semantics $\Vdash$ would not enjoy monotonicity. Take, for example, model $\mathscr{C}'$ from Fig. 1.iii reprinted here, where we give name X' to the edge 0_a–0_b.



Note that $\mathscr{C}', X' \Vdash \neg p_c$ whereas $\mathscr{C}', Y' \Vdash p_c$. This simultaneously shows the lack of upwards and downwards monotonicity for all four of the languages we consider.

Unsurprisingly, this infidelity translates to the real logical differences between the two-valued facet- and face-semantics, in contrast to Theorem 10 above.

Proposition 14. *$\Vdash$-validity is different from $\Vdash^{\mathscr{F}}$-validity.*

Proof. Consider the formula $\varphi = \widehat{K}_a\top \to \widehat{K}_a\neg p_b$ where $a, b \in A$ are two distinct agents. We show that $\nVdash^{\mathscr{F}} \varphi$ whereas $\Vdash \varphi$, distinguishing the two validities.

To show that $\nVdash^{\mathscr{F}} \widehat{K}_a\top \to \widehat{K}_a\neg p_b$, consider the following model $\mathscr{C}^-$ with the only facet being X:

$$\mathscr{C}^- : \quad 0_a \overset{X}{\text{———}} 1_b$$

Since $a \in \chi(X)$, we have $\mathscr{C}^-, X \Vdash^{\mathscr{F}} \widehat{K}_a\top$. But $\mathscr{C}^-, X \nVdash^{\mathscr{F}} \neg p_b$, meaning that $\mathscr{C}^-, X \nVdash^{\mathscr{F}} \widehat{K}_a\neg p_b$. Thus, overall, $\mathscr{C}^-, X \nVdash^{\mathscr{F}} \widehat{K}_a\top \to \widehat{K}_a\neg p_b$.

On the other hand, $\Vdash \widehat{K}_a\top \to \widehat{K}_a\neg p_b$ for the simple reason that any face containing an a-vertex makes $\neg p_b$ true in that vertex. Indeed, let X be an arbitrary face of an arbitrary model $\mathscr{C} = (C, \chi, \ell)$ such that $\mathscr{C}, X \Vdash \widehat{K}_a\top$. Then $\chi(v) = a$ for some vertex $v \in X$. Since $b \notin \chi(v)$, we are guaranteed that $p_b \notin \ell(v)$, which yields $\mathscr{C}, v \nVdash p_b$ and $\mathscr{C}, v \Vdash \neg p_b$. Finally, since $a \in \chi(X \cap \{v\})$, we have $\mathscr{C}, X \Vdash \widehat{K}_a\neg p_b$. Since $(\mathscr{C}, X)$ was arbitrary, we conclude that $\Vdash \widehat{K}_a\top \to \widehat{K}_a\neg p_b$. □

Proposition 14 shows that the two-valued face-semantics is infelicitous for impure complexes. You really do not want to have $\widehat{K}_a\top \to \widehat{K}_a\neg p_b$ as a theorem. Especially since it creates an asymmetry of local truth values in light of $\nVdash \widehat{K}_a\top \to \widehat{K}_a p_b$ (for which a singleton a-colored node is a countermodel).

We will therefore, from here on, for the two-valued case consider only the facet-semantics.

Lemma 15. $\Vdash^{\mathscr{F}} a \leftrightarrow \widehat{K}_a\top$.

Proof. Since all formulas are defined, it is sufficient to show that $\mathscr{C}, X \Vdash^{\mathscr{F}} a$ iff $\mathscr{C}, X \Vdash^{\mathscr{F}} \widehat{K}_a\top$ for any facet $X \in \mathscr{F}(C)$ of any simplicial model $\mathscr{C} = (C, \chi, \ell)$. Given that $\mathscr{C}, X \Vdash^{\mathscr{F}} \top$, we have the following equivalences: $\mathscr{C}, X \Vdash^{\mathscr{F}} a$ iff $a \in \chi(X)$ and, further,

$$a \in \chi(X) \iff a \in \chi(X \cap X) \iff a \in \chi(X \cap X) \text{ and } \mathscr{C}, X \Vdash^{\mathscr{F}} \top \iff \mathscr{C}, X \Vdash^{\mathscr{F}} \widehat{K}_a\top \quad (4)$$

□

This lemma means that in the two-valued case global atoms are expressible already in $\mathscr{L}_K^{\mathsf{loc}}$. Thus, for the two-valued (facet) semantics $\Vdash^{\mathscr{F}}$, we can restrict ourselves to the language $\mathscr{L}_D^{\mathsf{loc}}$ (or $\mathscr{L}_K^{\mathsf{loc}}$) without the loss of expressivity.[3] We will from here on only consider language $\mathscr{L}_D^{\mathsf{loc}}$ for the two-valued semantics.

[3]This result is not so unlike redefining a propositional variable *correct$_a$*, stating that agent a is correct, as $\neg H_a\bot$ in [7], where H_a is the hope modality.

Remark 16. Note that $\Vdash\!\vdash^{\mathscr{F}} a \leftrightarrow \widehat{K}_a\top$ also for the three-valued semantics but there $\top$ is only expressible in the facet-semantics and in presence of global atoms (see Remark 11). Hence, replacing a with $\widehat{K}_a\top$ would not remove global atoms from the language. Indeed, as already mentioned, global atoms are not three-valued-expressible in $\mathscr{L}_D^{\mathsf{loc}}$ because no formula of $\mathscr{L}_D^{\mathsf{loc}}$ is defined in all facets.

4 Translating three-valued into two-valued semantics

We provide a translation from language $\mathscr{L}_D^{\mathsf{gloc}}$ into $\mathscr{L}_D^{\mathsf{loc}}$. It consists of two parts. For any formula $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$ we define by mutual recursion

- formula $\varphi^{\bowtie} \in \mathscr{L}_D^{\mathsf{loc}}$ that determines whether φ is *defined* in some given $(\mathscr{C}, X)$ and
- formula $\varphi^{\sharp} \in \mathscr{L}_D^{\mathsf{loc}}$ that determines whether a defined formula φ is *true* in that $(\mathscr{C}, X)$.

This covers all our tracks in three-valued semantics, as there φ may be

- undefined, in which case $\varphi^{\bowtie}$ is false and, as we will see, so is $\varphi^{\sharp}$;
- true, in which case both $\varphi^{\bowtie}$ and $\varphi^{\sharp}$ must be true;
- false, in which case $\varphi^{\bowtie}$ is true but $\varphi^{\sharp}$ is false.

The translation from $\mathscr{L}_D^{\mathsf{gloc}}$ to $\mathscr{L}_D^{\mathsf{loc}}$ also determines one from $\mathscr{L}_D^{\mathsf{loc}}$ to $\mathscr{L}_D^{\mathsf{loc}}$, by removing the $a^{\bowtie}$ and $a^{\sharp}$ clauses.

Definition 17 (Translations)**.**

$$\begin{array}{lcllcl}
a^{\bowtie} & := & \top & a^{\sharp} & := & \widehat{K}_a\top \\
p_a^{\bowtie} & := & \widehat{K}_a\top & p_a^{\sharp} & := & p_a \\
(\neg\varphi)^{\bowtie} & := & \varphi^{\bowtie} & (\neg\varphi)^{\sharp} & := & (\neg\varphi)^{\bowtie} \wedge \neg\varphi^{\sharp} \\
(\varphi\wedge\psi)^{\bowtie} & := & \varphi^{\bowtie}\wedge\psi^{\bowtie} & (\varphi\wedge\psi)^{\sharp} & := & \varphi^{\sharp}\wedge\psi^{\sharp} \\
(\widehat{D}_B\varphi)^{\bowtie} & := & \widehat{D}_B\varphi^{\bowtie} & (\widehat{D}_B\varphi)^{\sharp} & := & \widehat{D}_B\varphi^{\sharp}
\end{array}$$

The main result to prove here is as follows.

Theorem 18. *For any model $\mathscr{C} = (C, \chi, \ell)$, facet $X \in \mathscr{F}(C)$, and formula $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$*

$$\mathscr{C}, X \bowtie^{\mathscr{F}} \varphi \iff \mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^{\bowtie}; \tag{5}$$

$$\mathscr{C}, X \Vdash\!\vdash^{\mathscr{F}} \varphi \iff \mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^{\sharp}. \tag{6}$$

Proof. We prove both statements by mutual induction on $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$:

Case a Here $a^{\bowtie} = \top$ and $a^{\sharp} = \widehat{K}_a\top$. For (5), both $\mathscr{C}, X \bowtie^{\mathscr{F}} a$ and $\mathscr{C}, X \Vdash^{\mathscr{F}} \top$. For (6),

$$\mathscr{C}, X \Vdash\!\vdash^{\mathscr{F}} a \quad\Leftrightarrow\quad a \in \chi(X) \quad\overset{(4)}{\iff}\quad \mathscr{C}, X \Vdash^{\mathscr{F}} \widehat{K}_a\top \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} a^{\sharp}.$$

Case p_a Here $p_a^{\bowtie} = \widehat{K}_a\top$ and $p_a^{\sharp} = p_a$. For (6), the statement is trivial since the three-valued and two-valued definitions of satisfaction coincide for p_a. For (5),

$$\mathscr{C}, X \bowtie^{\mathscr{F}} p_a \quad\Leftrightarrow\quad a \in \chi(X) \quad\overset{(4)}{\iff}\quad \mathscr{C}, X \Vdash^{\mathscr{F}} \widehat{K}_a\top \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} p_a^{\bowtie}.$$

Case $\neg\varphi$ Here $(\neg\varphi)^{\bowtie} = \varphi^{\bowtie}$ and $(\neg\varphi)^{\sharp} = (\neg\varphi)^{\bowtie} \wedge \neg\varphi^{\sharp}$. For (5), the statement follows by IH(5) since $\mathscr{C}, X \bowtie^{\mathscr{F}} \neg\varphi$ iff $\mathscr{C}, X \bowtie^{\mathscr{F}} \varphi$. Using that, for (6),

$$\mathscr{C}, X \Vdash\!\vdash^{\mathscr{F}} \neg\varphi \quad\Leftrightarrow\quad \mathscr{C}, X \bowtie^{\mathscr{F}} \neg\varphi \text{ and } \mathscr{C}, X \not\Vdash\!\vdash^{\mathscr{F}} \varphi \quad\overset{(5),\mathrm{IH}(6)}{\iff}$$

$$\mathscr{C}, X \Vdash^{\mathscr{F}} (\neg\varphi)^{\bowtie} \text{ and } \mathscr{C}, X \nVdash^{\mathscr{F}} \varphi^{\sharp} \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} (\neg\varphi)^{\bowtie} \text{ and } \mathscr{C}, X \Vdash^{\mathscr{F}} \neg\varphi^{\sharp} \quad\Leftrightarrow$$

$$\mathscr{C}, X \Vdash^{\mathscr{F}} (\neg\varphi)^{\bowtie} \wedge \neg\varphi^{\sharp} \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} (\neg\varphi)^{\sharp}.$$

Cases $\varphi \wedge \psi$ and $\widehat{D}_B\varphi$ Here $(\varphi \wedge \psi)^\dagger = \varphi^\dagger \wedge \psi^\dagger$ and $(\widehat{D}_B\varphi)^\dagger = \widehat{D}_B\varphi^\dagger$ for $\dagger \in \{\bowtie, \sharp\}$. Because the $\bowtie$- and $\sharp$-translations work the same way in both cases, the arguments for (5) and (6) are analogous. We present the proof of the former only for $\varphi \wedge \psi$ and of the latter only for $\widehat{D}_B\varphi$:

$$\mathscr{C}, X \bowtie^{\mathscr{F}} \varphi \wedge \psi \quad\Leftrightarrow\quad \mathscr{C}, X \bowtie^{\mathscr{F}} \varphi \text{ and } \mathscr{C}, X \bowtie^{\mathscr{F}} \psi \quad \xLeftrightarrow{\text{IH(5)}}$$

$$\mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^{\bowtie} \text{ and } \mathscr{C}, X \Vdash^{\mathscr{F}} \psi^{\bowtie} \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^{\bowtie} \wedge \psi^{\bowtie} \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} (\varphi \wedge \psi)^{\bowtie};$$

$$\mathscr{C}, X \Vvdash^{\mathscr{F}} \widehat{D}_B\varphi \quad\Leftrightarrow\quad \mathscr{C}, Y \Vvdash^{\mathscr{F}} \varphi \text{ for some } Y \in \mathscr{F}(C) \text{ with } B \subseteq \chi(X \cap Y) \quad \xLeftrightarrow{\text{IH(6)}}$$

$$\mathscr{C}, Y \Vdash^{\mathscr{F}} \varphi^\sharp \text{ for some } Y \in \mathscr{F}(C) \text{ with } B \subseteq \chi(X \cap Y) \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} \widehat{D}_B\varphi^\sharp \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} (\widehat{D}_B\varphi)^\sharp. \quad \Box$$

By omitting the first clause in the inductive proof above, we can conclude that Theorem 18 also holds for the local language $\mathscr{L}_D^{\mathsf{loc}}$.

We can also represent the non-standard notion of three-valued validity in two-valued semantics:

Corollary 19. $\Vvdash \varphi \quad\Leftrightarrow\quad \Vvdash^{\mathscr{F}} \varphi \quad\Leftrightarrow\quad \Vdash^{\mathscr{F}} \varphi^{\bowtie} \to \varphi^\sharp \qquad$ *for any* $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$.

Proof. It follows from Def. 3 and Theorems 10 and 18. □

It should also be noted that the two- and three-valued semantics coincide on pure simplicial models. In fact, this agreement can serve as an independent objective distinction between pure and impure models:

Corollary 20. *For any* <u>*pure*</u> *simplicial model* $\mathscr{C} = (C, \chi, \ell)$*, we have* $\mathscr{C}, X \bowtie^{\mathscr{F}} \varphi$ *and*

$$\mathscr{C}, X \Vvdash^{\mathscr{F}} \varphi \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} \varphi \qquad \textit{for any facet } X \in \mathscr{F}(C) \textit{ and formula } \varphi \in \mathscr{L}_D^{\mathsf{gloc}}. \tag{7}$$

Proof. It is easy to show by induction on the construction of φ that $\mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^{\bowtie}$. Indeed, for atoms both $\top$ and $a = \widehat{K}_a\top$ are true in every facet X of the pure complex $\mathscr{C}$; similarly $B \subseteq A = \chi(X \cap X)$ ensures the modal clause. The first statement now follows from (5). In view of this, a simple induction argument shows that for pure models the $\sharp$-translation can be pushed through all connectives and eventually removed completely: $\mathscr{C}, X \Vdash^{\mathscr{F}} \varphi \leftrightarrow \varphi^\sharp$. Thus, the second statement follows from (6). □

In fact, (7) can be viewed as an alternative, functional definition of pure models.

Theorem 21. *A simplicial model* $\mathscr{C} = (C, \chi, \ell)$ *is pure iff* (7) *holds.*

Proof. The only-if-direction is proved in Corollary 20. For the if-direction, by contraposition, assume $\mathscr{C}$ is not pure, i.e., there is a facet $X \in \mathscr{F}(C)$ and agent $a \in A$ such that $a \notin \chi(X)$. We, therefore, have $\mathscr{C}, X \not\Vvdash^{\mathscr{F}} \neg p_a$ because $\mathscr{C}, X \not\bowtie^{\mathscr{F}} \neg p_a$ while, at the same time, $\mathscr{C}, X \Vdash^{\mathscr{F}} \neg p_a$ because $p_a \notin \ell(X)$ in violation of (7). □

Example 22. For the simplicial model $\mathscr{C}$ in Fig. 1.ix, we have $\mathscr{C} \Vvdash^{\mathscr{F}} K_a p_b \wedge K_a p_c$, while, at the same time, $\mathscr{C} \Vdash^{\mathscr{F}} \neg K_a p_b \wedge \neg K_a p_c$. This disagreement of the two semantics is why the model in Fig. 1.ix should not be considered pure, despite all its facets having the same dimension 1.

To conclude this section, we give some examples of the translation, and a number of derived propositions that might further throw some intuitive light on this translation (where we note once more that all these are also valid for the language $\mathscr{L}_D^{\mathsf{loc}}$).

Example 23. It is easy to see that $(\neg p_a)^\sharp = \widehat{K}_a\top \wedge \neg p_a$ and $(\neg a)^\sharp = \top \wedge \neg\widehat{K}_a\top$, which, modulo abbreviations and two-valued equivalences, yields $\Vdash^{\mathscr{F}} (\neg p_a)^\sharp \leftrightarrow a \wedge \neg p_a$ and $\Vdash^{\mathscr{F}} (\neg a)^\sharp \leftrightarrow \neg a$.

Consider two agents $a, b \in A$, a simplicial model $\mathscr{C} = (C, \chi, \ell)$, and its facet $X \in \mathscr{F}(C)$.

$$(K_a p_b)^\bowtie = (\neg\widehat{K}_a\neg p_b)^\bowtie = (\widehat{K}_a\neg p_b)^\bowtie = \widehat{K}_a(\neg p_b)^\bowtie = \widehat{K}_a p_b^\bowtie = \widehat{K}_a\widehat{K}_b\top;$$

$$\begin{aligned}(K_a p_b)^\sharp = (\neg\widehat{K}_a\neg p_b)^\sharp = (\neg\widehat{K}_a\neg p_b)^\bowtie \wedge \neg(\widehat{K}_a\neg p_b)^\sharp = \widehat{K}_a\widehat{K}_b\top \wedge \neg\widehat{K}_a(\neg p_b)^\sharp = \\ \widehat{K}_a\widehat{K}_b\top \wedge \neg\widehat{K}_a\left((\neg p_b)^\bowtie \wedge \neg p_b^\sharp\right) = \widehat{K}_a\widehat{K}_b\top \wedge \neg\widehat{K}_a(p_b^\bowtie \wedge \neg p_b) = \widehat{K}_a\widehat{K}_b\top \wedge \neg\widehat{K}_a\left(\widehat{K}_b\top \wedge \neg p_b\right).\end{aligned}$$

Since $\Vdash^{\mathscr{F}} b \leftrightarrow \widehat{K}_b\top$ by Lemma 15 and $\Vdash^{\mathscr{F}} \widehat{K}_b\top \wedge \neg p_b \leftrightarrow \neg(\widehat{K}_b\top \to p_b)$, we conclude that

$$\mathscr{C}, X \Vvdash^{\mathscr{F}} K_a p_b \quad\Longleftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} \widehat{K}_a b \wedge K_a(b \to p_b).$$

This is what we want: for $K_a p_b$ to be true, a should consider it possible that b is alive, and for all facets considered possible by a where b is alive, p_b should be true. In particular, for $K_a p_b$ to be true it is not necessary that b be actually alive.

From this point on, we will routinely abbreviate $\widehat{K}_b\top$ as b without leaving language $\mathscr{L}_D^{\mathsf{loc}}$.

Proposition 24. $\Vdash^{\mathscr{F}} \varphi^\sharp \to \varphi^\bowtie$ *for any* $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$.

Proof. It is sufficient to prove, by induction on the formula structure, that for all models $\mathscr{C} = (C, \chi, \ell)$ and all facets $X \in \mathscr{F}(C)$, if $\mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^\sharp$, then $\mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^\bowtie$. Since $a^\bowtie = \top$, the statement is trivial for global atoms. For local atoms, with $p_a^\sharp = p_a$ and $p_a^\bowtie = a$,

$$\mathscr{C}, X \Vdash^{\mathscr{F}} p_a \quad\Leftrightarrow\quad p_a \in \ell(X) \quad\Rightarrow\quad a \in \chi(X) \quad\overset{(4)}{\Longleftrightarrow}\quad \mathscr{C}, X \Vdash^{\mathscr{F}} a.$$

For $\neg\varphi$, the statement follows directly from the definition of $\sharp$. The remaining two cases easily follow by IH. We only show the case of $\varphi \wedge \psi$:

$$\begin{aligned}\mathscr{C}, X \Vdash^{\mathscr{F}} (\varphi \wedge \psi)^\sharp \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^\sharp \wedge \psi^\sharp \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^\sharp \text{ and } \mathscr{C}, X \Vdash^{\mathscr{F}} \psi^\sharp \quad\overset{\text{IH}}{\Longrightarrow} \\ \mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^\bowtie \text{ and } \mathscr{C}, X \Vdash^{\mathscr{F}} \psi^\bowtie \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} \varphi^\bowtie \wedge \psi^\bowtie \quad\Leftrightarrow\quad \mathscr{C}, X \Vdash^{\mathscr{F}} (\varphi \wedge \psi)^\bowtie. \quad \square\end{aligned}$$

Proposition 25. $\Vdash^{\mathscr{F}} (\neg\neg\varphi)^\sharp \leftrightarrow \varphi^\sharp$ *for any* $\varphi \in \mathscr{L}_D^{\mathsf{gloc}}$.

Proof. By definition, we have

$$(\neg\neg\varphi)^\sharp = (\neg\neg\varphi)^\bowtie \wedge \neg(\neg\varphi)^\sharp = (\neg\varphi)^\bowtie \wedge \neg((\neg\varphi)^\bowtie \wedge \neg\varphi^\sharp) = \varphi^\bowtie \wedge \neg(\varphi^\bowtie \wedge \neg\varphi^\sharp).$$

By standard propositional reasoning, $\Vdash^{\mathscr{F}} \left(\varphi^\bowtie \wedge \neg(\varphi^\bowtie \wedge \neg\varphi^\sharp)\right) \leftrightarrow \left(\varphi^\bowtie \wedge \neg\neg\varphi^\sharp\right)$. Therefore, we have $\Vdash^{\mathscr{F}} (\neg\neg\varphi)^\sharp \leftrightarrow \varphi^\bowtie \wedge \varphi^\sharp$. The desired statement now follows from Prop. 24. $\square$

5 Discussion and conclusion

In this paper, we analyzed and compared four different logical languages for impure simplicial complexes and four semantics for them: two two-valued and two three-valued epistemic semantics. Our main findings can be summarized as follows:

- The two-valued face-semantics $\Vdash$ is infelicitous.
- The three-valued facet-semantics $\Vvdash^{\mathscr{F}}$ and face-semantics $\Vvdash$ produce the same logic and, hence, can be used interchangeably.
- We provided a faithful embedding from the three-valued facet semantics $\Vvdash^{\mathscr{F}}$ into the two-valued facet semantics $\Vdash^{\mathscr{F}}$.
- Global propositional variables describing whether agents are alive or dead increase the expressivity of the language in the three-valued case, but not in the two-valued one.
- The two-valued facet-semantics $\Vdash^{\mathscr{F}}$ and the three-valued facet-semantics $\Vvdash^{\mathscr{F}}$ coincide on pure simplicial models.

By relating three-valued semantics to two-valued semantics for impure complexes in a purely technical way, we hope we have filled a gap between publications like [21, 19] on the one hand and publications like [12, 31] on the other. Clearly, something different is going on here, but what is it exactly? Concerning the truth values, we provided the answer. However, let us elaborate on the other differences between such approaches. The most striking of them is that the impure complexes of [21, 19] do not have local propositional variables for the agents (processes). Valuations do not apply to vertices. Instead, valuations apply to facets only. This is best explained by an example:

Reconsider Fig. 1. In the approach of [21, 19], the modeler has to choose whether the value of p_c in X of Fig. 1.vii is false or true, and, therefore, whether the "original complex" before process c became inactive, was Fig. 1.i or viii. In the underlying contribution and in [12, 31] this choice is not made and left open. One could therefore consider Fig. 1.vii as some kind of quotient of Figs. 1.i and viii following a crash. The choice made in [21, 19] is essential in order to still allow arbitrary values for processes and keep it possible that agents have positive knowledge. Their two-valued semantics for knowledge is a special case of the two-valued semantics given in Def. 12: $\mathscr{C},X \Vdash^{\mathscr{F}} K_a\varphi$ iff $\mathscr{C},Y \Vdash^{\mathscr{F}} \varphi$ for all $Y \in \mathscr{F}(C)$ with $a \in \chi(X \cap Y)$. Applied to Fig. 1.vii we can then only justify that $\mathscr{C},X \Vdash^{\mathscr{F}} K_a p_c$ if $\mathscr{C},X \Vdash^{\mathscr{F}} p_c$ and $\mathscr{C},Y \Vdash^{\mathscr{F}} p_c$, in other words, if the bogus valuation of p_c in X made it true there. Otherwise, a does not know the value of p_c.

As shown, the two-valued face semantics is even infelicitous for *pure* complexes, already for the simple reason that an atom p_a is false in a face X whenever a is not a color in X, but then 'becomes' true if X is contained in a facet Y where p_a labels the a vertex. This may suggest an insuperable problem but, not surprisingly, there are yet more different two-valued face semantics (that also differ from [21, 19]). To interpret formulas in faces that are not facets we can also use the multi-pointed semantics of [9, Sect. 'Local semantics for simplicial complexes'], wherein it is defined that $\mathscr{C},\boldsymbol{X} \Vdash \varphi$ for a set $\boldsymbol{X}$ of facets, iff $\mathscr{C},X \Vdash \varphi$ for all $X \in \boldsymbol{X}$. In particular, now consider the set of facets containing a face X. For a face X that is not a facet we then have that $\mathscr{C},X \Vdash \varphi$ iff $\mathscr{C},Y \Vdash \varphi$ for (the set of) all facets Y containing X. (In general, we can even define for arbitrary faces that $\mathscr{C},X \Vdash \varphi$ iff $\mathscr{C},\mathsf{star}(X) \Vdash \varphi$, where $\mathsf{star}(X) = \{Y \in C \mid X \subseteq Y\}$.) Consequently, in such an approach we would have that in the vertex 0_a of Fig. 1.i atom p_c is true (because it is true in both facets) whereas in the vertex also named 0_a of Fig. 1.viii atom p_c is false. Multi-pointed semantics are common fare in Kripke model settings, in particular for model checking applications and in dynamics [14].

For further research, we wish to generalize our setting from simplicial complexes to (semi-)simplicial sets, and, correspondingly, from standard multi-agent Kripke models to Kripke models where each group $B \subseteq A$ of agents has its own associated equivalence relation $\sim_B$ and where the agents in B together may know more than the agents in B separately, even when merging their knowledge. In other words, we may then have that $\sim_B$ is strictly contained in $\bigcap_{b \in B} \sim_b$. In modal logic, such models seemed a rather technical tool so far, merely complicating the construction of canonical models, in works as [21]. But in combinatorial topology, scenarios where a whole is more than the sum of its parts are very natural, as amply shown in [19].

Another direction of further research would be the incorporation of dynamics such as in protocols.

References

[1] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (2002): *Alternating-Time Temporal Logic*. *Journal of the ACM* 49(5), pp. 672–713, doi:`10.1145/585265.585270`.

[2] Philippe Balbiani, Hans van Ditmarsch & Saúl Fernández González (2022): *Asynchronous Announcements*. *ACM Transactions on Computational Logic* 23(2):10, doi:`10.1145/3481806`.

[3] Alexandru Baltag, Lawrence S. Moss & Sławomir Solecki (1998): *The Logic of Public Announcements, Common Knowledge, and Private Suspicions*. In Itzhak Gilboa, editor: *Theoretical Aspects of Rationality and Knowledge: Proceedings of the Seventh Conference (TARK 1998)*, Morgan Kaufmann, pp. 43–56. Available at `http://tark.org/proceedings/tark_jul22_98/p43-baltag.pdf`.

[4] Johan van Benthem, Jelle Gerbrandy, Tomohiro Hoshi & Eric Pacuit (2009): *Merging Frameworks for Interaction*. *Journal of Philosophical Logic* 38(5), pp. 491–526, doi:`10.1007/s10992-008-9099-x`.

[5] Ofer Biran, Shlomo Moran & Shmuel Zak (1990): *A combinatorial characterization of the distributed 1-solvable tasks*. *Journal of Algorithms* 11(3), pp. 420–440, doi:`10.1016/0196-6774(90)90020-F`.

[6] Cédric Dégremont, Benedikt Löwe & Andreas Witzel (2011): *The Synchronicity of Dynamic Epistemic Logic*. In Krzysztof R. Apt, editor: *TARK XIII, Theoretical Aspects of Rationality and Knowledge: Proceedings of the Thirteenth Conference (TARK 2011)*, Association for Computing Machinery, pp. 145–152, doi:`10.1145/2000378.2000395`.

[7] Hans van Ditmarsch, Krisztina Fruzsa & Roman Kuznets (2022): *A New Hope*. In David Fernández-Duque, Alessandra Palmigiano & Sophie Pinchinat, editors: *Advances in Modal Logic*, 14, College Publications, pp. 349–369.

[8] Hans van Ditmarsch, Éric Goubault, Marijana Lazić, Jérémy Ledent & Sergio Rajsbaum (2021): *A dynamic epistemic logic analysis of equality negation and other epistemic covering tasks*. *Journal of Logical and Algebraic Methods in Programming* 121:100662, doi:`10.1016/j.jlamp.2021.100662`.

[9] Hans van Ditmarsch, Éric Goubault, Jérémy Ledent & Sergio Rajsbaum (2022): *Knowledge and Simplicial Complexes*. In Björn Lundgren & Nancy Abigail Nuñez Hernández, editors: *Philosophy of Computing: Themes from IACAP 2019*, *Philosophical Studies Series* 143, Springer, pp. 1–50, doi:`10.1007/978-3-030-75267-5_1`.

[10] Hans van Ditmarsch, Joseph Y. Halpern, Wiebe van der Hoek & Barteld Kooi (2015): *An Introduction to Logics of Knowledge and Belief*. In Hans van Ditmarsch, Joseph Y. Halpern, Wiebe van der Hoek & Barteld Kooi, editors: *Handbook of Epistemic Logic*, College Publications, pp. 1–51.

[11] Hans van Ditmarsch, Wiebe van der Hoek & Barteld Kooi (2007): *Dynamic Epistemic Logic*. *Synthese Library* 337, Springer, doi:`10.1007/978-1-4020-5839-4`.

[12] Hans van Ditmarsch & Roman Kuznets (2023): *Wanted Dead or Alive: Epistemic logic for impure simplicial complexes*. Eprint 2103.03032, arXiv, doi:`10.48550/arXiv.2103.03032`. Accepted to *Journal of Logic and Computation*.

[13] Clare Dixon, Cláudia Nalon & Ram Ramanujam (2015): *Knowledge and Time*. In Hans van Ditmarsch, Joseph Y. Halpern, Wiebe van der Hoek & Barteld Kooi, editors: *Handbook of Epistemic Logic*, College Publications, pp. 205–259.

[14] Jan van Eijck (2007): *DEMO — A Demo of Epistemic Modelling*. In Johan van Benthem, Dov Gabbay & Benedikt Löwe, editors: *Interactive Logic: Selected Papers from the 7th Augustus de Morgan Workshop, London*, *Texts in Logic and Games* 1, Amsterdam University Press, pp. 303–362. Available at `https://www.jstor.org/stable/j.ctt45kdbf.15`.

[15] Ronald Fagin, Joseph Y. Halpern, Yoram Moses & Moshe Y. Vardi (1995): *Reasoning About Knowledge*. MIT Press, doi:10.7551/mitpress/5803.001.0001.

[16] Michael J. Fischer, Nancy A. Lynch & Michael S. Paterson (1985): *Impossibility of Distributed Consensus with One Faulty Process*. *Journal of the ACM* 32(2), pp. 374–382, doi:10.1145/3149.214121.

[17] Krisztina Fruzsa, Roman Kuznets & Ulrich Schmid (2021): *Fire!* In Joseph Halpern & Andrés Perea, editors: *Proceedings Eighteenth Conference on Theoretical Aspects of Rationality and Knowledge, Beijing, China, June 25–27, 2021*, *Electronic Proceedings in Theoretical Computer Science* 335, Open Publishing Association, pp. 139–153, doi:10.4204/EPTCS.335.13.

[18] Jelle Gerbrandy & Willem Groeneveld (1997): *Reasoning about Information Change*. *Journal of Logic, Language, and Information* 6(2), pp. 147–169, doi:10.1023/A:1008222603071.

[19] Éric Goubault, Roman Kniazev, Jérémy Ledent & Sergio Rajsbaum (2023): *Semi-simplicial Set Models for Distributed Knowledge*. In: *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), 26–29 June 2023, Boston, USA*, IEEE, doi:10.1109/LICS56636.2023.10175737.

[20] Éric Goubault, Jérémy Ledent & Sergio Rajsbaum (2021): *A simplicial complex model for dynamic epistemic logic to study distributed task computability*. *Information and Computation* 278:104597, doi:10.1016/j.ic.2020.104597.

[21] Éric Goubault, Jérémy Ledent & Sergio Rajsbaum (2022): *A Simplicial Model for* $\mathsf{KB4}_n$*: Epistemic Logic with Agents That May Die*. In Petra Berenbrink & Benjamin Monmege, editors: *39th International Symposium on Theoretical Aspects of Computer Science: STACS 2022, March 15–18, 2022, Marseille, France (Virtual Conference)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 219, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 33:1–33:20, doi:10.4230/LIPIcs.STACS.2022.33.

[22] Joseph Y. Halpern & Yoram Moses (1990): *Knowledge and Common Knowledge in a Distributed Environment*. *Journal of the ACM* 37(3), pp. 549–587, doi:10.1145/79147.79161.

[23] Maurice Herlihy, Dmitry Kozlov & Sergio Rajsbaum (2014): *Distributed Computing through Combinatorial Topology*. Morgan Kaufmann, doi:10.1016/C2011-0-07032-1.

[24] Maurice Herlihy & Nir Shavit (1999): *The Topological Structure of Asynchronous Computability*. *Journal of the ACM* 46(6), pp. 858–923, doi:10.1145/331524.331529.

[25] Jaakko Hintikka (1962): *Knowledge and Belief: An Introduction to the Logic of the Two Notions*. Cornell University Press.

[26] Jérémy Ledent (2019): *Geometric semantics for asynchronous computability*. Ph.D. thesis, Paris-Saclay University, Palaiseau, France. Available at https://theses.hal.science/tel-02445180. Prepared at École polytechnique.

[27] Michael C. Loui & Hosame H. Abu-Amara (1987): *Memory Requirements for Agreement among Unreliable Asynchronous Processes*. In Franco P. Preparata, editor: *Parallel and Distributed Computing*, *Advances in Computing Research: A Research Annual* 4, JAI Press, pp. 163–183.

[28] Lawrence S. Moss (2015): *Dynamic Epistemic Logic*. In Hans van Ditmarsch, Joseph Y. Halpern, Wiebe van der Hoek & Barteld Kooi, editors: *Handbook of Epistemic Logic*, College Publications, pp. 261–312.

[29] Daniel Pfleger & Ulrich Schmid (2018): *On Knowledge and Communication Complexity in Distributed Systems*. In Zvi Lotker & Boaz Patt-Shamir, editors: *Structural Information and Communication Complexity: 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18–21, 2018, Revised Selected Papers*, *Lecture Notes in Computer Science* 11085, Springer, pp. 312–330, doi:10.1007/978-3-030-01325-7_27.

[30] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: *18th Annual Symposium on Foundations of Computer Science*, IEEE, pp. 46–57, doi:10.1109/SFCS.1977.32.

[31] Rojo Randrianomentsoa, Hans van Ditmarsch & Roman Kuznets (2023): *Impure Simplicial Complexes: Complete Axiomatization*. *Logical Methods in Computer Science*, doi:10.48550/arXiv.2211.13543. In press.

[32] Floris Roelofsen (2007): *Distributed knowledge*. *Journal of Applied Non-Classical Logics* 17(2), pp. 255–273, doi:10.3166/jancl.17.255-273.

[33] Diego A. Velázquez, Armando Castañeda & David A. Rosenblueth (2021): *Communication Pattern Models: An Extension of Action Models for Dynamic-Network Distributed Systems*. In Joseph Halpern & Andrés Perea, editors: *Proceedings Eighteenth Conference on Theoretical Aspects of Rationality and Knowledge, Beijing, China, June 25–27, 2021*, *Electronic Proceedings in Theoretical Computer Science* 335, Open Publishing Association, pp. 307–321, doi:10.4204/EPTCS.335.29.

Modal Logic Characterizations of Forward, Reverse, and Forward-Reverse Bisimilarities

Marco Bernardo   Andrea Esposito
Dipartimento di Scienze Pure e Applicate, Università di Urbino, Urbino, Italy

Reversible systems feature both forward computations and backward computations, where the latter undo the effects of the former in a causally consistent manner. The compositionality properties and equational characterizations of strong and weak variants of forward-reverse bisimilarity as well as of its two components, i.e., forward bisimilarity and reverse bisimilarity, have been investigated on a minimal process calculus for nondeterministic reversible systems that are sequential, so as to be neutral with respect to interleaving vs. truly concurrent semantics of parallel composition. In this paper we provide logical characterizations for the considered bisimilarities based on forward and backward modalities, which reveals that strong and weak reverse bisimilarities respectively correspond to strong and weak reverse trace equivalences. Moreover, we establish a clear connection between weak forward-reverse bisimilarity and branching bisimilarity, so that the former inherits two further logical characterizations from the latter over a specific class of processes.

1 Introduction

Reversibility in computing started to gain attention since the seminal works [13, 2], where it was shown that reversible computations may achieve low levels of heat dissipation. Nowadays *reversible computing* has many applications ranging from computational biochemistry and parallel discrete-event simulation to robotics, control theory, fault tolerant systems, and concurrent program debugging.

In a reversible system, two directions of computation can be observed: a *forward* one, coinciding with the normal way of computing, and a *backward* one, along which the effects of the forward one are undone when needed in a *causally consistent* way, i.e., by returning to a past consistent state. The latter task is not easy to accomplish in a concurrent system, because the undo procedure necessarily starts from the last performed action and this may not be unique. The usually adopted strategy is that an action can be undone provided that all of its consequences, if any, have been undone beforehand [7].

In the process algebra literature, two approaches have been developed to reverse computations based on keeping track of past actions: the dynamic one of [7] and the static one of [18], later shown to be equivalent in terms of labeled transition systems isomorphism [14].

The former yields RCCS, a variant of CCS [16] that uses stack-based memories attached to processes to record all the actions executed by those processes. A single transition relation is defined, while actions are divided into forward and backward resulting in forward and backward transitions. This approach is suitable when the operational semantics is given in terms of reduction semantics, like in the case of very expressive calculi as well as programming languages.

In contrast, the latter proposes a general method, of which CCSK is a result, to reverse calculi, relying on the idea of retaining within the process syntax all executed actions, which are suitably decorated, and all dynamic operators, which are thus made static. A forward transition relation and a backward transition relation are separately defined, which are labeled with actions extended with communication keys so as to remember who synchronized with whom when going backward. This approach is very handy when it comes to deal with labeled transition systems and basic process calculi.

In [18] *forward-reverse bisimilarity* was introduced too. Unlike standard forward-only bisimilarity [17, 16], it is truly concurrent as it does not satisfy the expansion law of parallel composition into a choice among all possible action sequencings. The interleaving view can be restored in a reversible setting by employing *back-and-forth bisimilarity* [8]. This is defined on computation paths instead of states, thus preserving not only causality but also history as backward moves are constrained to take place along the path followed when going forward even in the presence of concurrency. In the latter setting, a single transition relation is considered, which is viewed as bidirectional, and in the bisimulation game the distinction between going forward or backward is made by matching outgoing or incoming transitions of the considered processes, respectively.

In [4] forward-reverse bisimilarity and its two components, i.e., forward bisimilarity and reverse bisimilarity, have been investigated in terms of compositionality properties and equational characterizations, both for nondeterministic processes and Markovian processes. In order to remain neutral with respect to interleaving view vs. true concurrency, the study has been conducted over a sequential processes calculus, in which parallel composition is not admitted so that not even the communication keys of [18] are needed. Furthermore, like in [8] a single transition relation has been defined and the distinction between outgoing and incoming transitions has been exploited in the bisimulation game. In [3] the investigation of compositionality and axiomatizations has been extended to weak variants of forward, reverse, and forward-reverse bisimilarities, i.e., variants that are capable of abstracting from unobservable actions, in the case of nondeterministic processes only.

In this paper we address the logical characterization of the aforementioned strong and weak bisimilarities over nondeterministic reversibile sequential processes. The objective is to single out suitable modal logics that induce equivalences that turn out to be alternative characterizations of the considered bisimilarities, so that two processes are bisimilar iff they satisfy the same set of formulas of the corresponding logic. Starting from Hennessy-Milner logic [11], which includes forward modalities whereby it is possible to characterize the standard forward-only strong and weak bisimilarities of [16], the idea is to add backward modalities in the spirit of [8] so as to be able to characterize reverse and forward-reverse strong and weak bisimilarities. Unlike [8], where back-and-forth bisimilarities as well as modality interpretations are defined over computation paths, in our reversible setting both the considered bisimilarities and the associated modal logic interpretations are defined over states.

Our study reveals that strong and weak reverse bisimilarities do not need conjunction in their logical characterizations. In other words, they boil down to strong and weak reverse trace equivalences, respectively. Moreover, recalling that branching bisimilarity [10] is known to coincide with weak back-and-forth bisimilarity defined over computation paths [8], we show that branching bisimilarity also coincides for a specific class of processes with our weak forward-reverse bisimilarity defined over states. Based on the results in [9], this opens the way to two further logical characterizations of the latter in addition to the one based on forward and backward modalities. The first characterization replaces the aforementioned modalities with an until operator, whilst the second one is given by the temporal logic CTL* without the next operator.

The paper is organized as follows. In Section 2 we recall syntax and semantics for the considered calculus of nondeterministic reversible sequential processes as well as the strong forward, reverse, and forward-reverse bisimilarities investigated in [4] and their weak counterparts examined in [3]. In Section 3 we provide the modal logic characterizations of all the aforementioned bisimilarities based on forward and backward modalities interpreted over states. In Section 4 we establish a clear connection between branching bisimilarity and our weak forward-reverse bisimilarity defined over states. In Section 5 we conclude with final remarks and directions for future work.

2 Background

2.1 Syntax of Nondeterministic Reversible Sequential Processes

Given a countable set A of actions – ranged over by a, b, c – including an unobservable action denoted by τ, the syntax of reversible sequential processes is defined as follows [4]:

$$P ::= \underline{0} \mid a\,.\,P \mid a^{\dagger}\,.\,P \mid P + P$$

where:

- $\underline{0}$ is the terminated process.
- $a\,.\,P$ is a process that can execute action a and whose forward continuation is P.
- $a^{\dagger}\,.\,P$ is a process that executed action a and whose forward continuation is inside P.
- $P_1 + P_2$ expresses a nondeterministic choice between P_1 and P_2 as far as both of them have not executed any action yet, otherwise only the one that was selected in the past can move.

We syntactically characterize through suitable predicates three classes of processes generated by the grammar above. Firstly, we have *initial* processes, i.e., processes in which all the actions are unexecuted:

$$\begin{array}{rcl} \mathit{initial}(\underline{0}) & & \\ \mathit{initial}(a\,.\,P) & \Longleftarrow & \mathit{initial}(P) \\ \mathit{initial}(P_1 + P_2) & \Longleftarrow & \mathit{initial}(P_1) \land \mathit{initial}(P_2) \end{array}$$

Secondly, we have *final* processes, i.e., processes in which all the actions along a single path have been executed:

$$\begin{array}{rcl} \mathit{final}(\underline{0}) & & \\ \mathit{final}(a^{\dagger}\,.\,P) & \Longleftarrow & \mathit{final}(P) \\ \mathit{final}(P_1 + P_2) & \Longleftarrow & (\mathit{final}(P_1) \land \mathit{initial}(P_2)) \lor \\ & & (\mathit{initial}(P_1) \land \mathit{final}(P_2)) \end{array}$$

Multiple paths arise only in the presence of alternative compositions. At each occurrence of $+$, only the subprocess chosen for execution can move, while the other one, although not selected, is kept as an initial subprocess within the overall process to support reversibility.

Thirdly, we have the processes that are *reachable* from an initial one, whose set we denote by $\mathbb{P}$:

$$\begin{array}{rcl} \mathit{reachable}(\underline{0}) & & \\ \mathit{reachable}(a\,.\,P) & \Longleftarrow & \mathit{initial}(P) \\ \mathit{reachable}(a^{\dagger}\,.\,P) & \Longleftarrow & \mathit{reachable}(P) \\ \mathit{reachable}(P_1 + P_2) & \Longleftarrow & (\mathit{reachable}(P_1) \land \mathit{initial}(P_2)) \lor \\ & & (\mathit{initial}(P_1) \land \mathit{reachable}(P_2)) \end{array}$$

It is worth noting that:

- $\underline{0}$ is the only process that is both initial and final as well as reachable.
- Any initial or final process is reachable too.
- $\mathbb{P}$ also contains processes that are neither initial nor final, like e.g. $a^{\dagger}\,.\,b\,.\,\underline{0}$.
- The relative positions of already executed actions and actions to be executed matter; in particular, an action of the former kind can never follow one of the latter kind. For instance, $a^{\dagger}\,.\,b\,.\,\underline{0} \in \mathbb{P}$ whereas $b\,.\,a^{\dagger}\,.\,\underline{0} \notin \mathbb{P}$.

$$(\textsc{Act}_{\mathrm{f}})\ \frac{\mathit{initial}(P)}{a\,.\,P \xrightarrow{a} a^{\dagger}\,.\,P} \qquad (\textsc{Act}_{\mathrm{p}})\ \frac{P \xrightarrow{b} P'}{a^{\dagger}\,.\,P \xrightarrow{b} a^{\dagger}\,.\,P'}$$

$$(\textsc{Cho}_{\mathrm{l}})\ \frac{P_1 \xrightarrow{a} P_1' \quad \mathit{initial}(P_2)}{P_1 + P_2 \xrightarrow{a} P_1' + P_2} \qquad (\textsc{Cho}_{\mathrm{r}})\ \frac{P_2 \xrightarrow{a} P_2' \quad \mathit{initial}(P_1)}{P_1 + P_2 \xrightarrow{a} P_1 + P_2'}$$

Table 1: Operational semantic rules for reversible action prefix and choice

2.2 Operational Semantic Rules

According to the approach of [18], dynamic operators such as action prefix and alternative composition have to be made static by the semantics, so as to retain within the syntax all the information needed to enable reversibility. For the sake of minimality, unlike [18] we do not generate two distinct transition relations – a forward one $\longrightarrow$ and a backward one $\leadsto$ – but a single transition relation, which we implicitly regard as being symmetric like in [8] to enforce the *loop property*: every executed action can be undone and every undone action can be redone.

In our setting, a backward transition from P' to P ($P' \overset{a}{\leadsto} P$) is subsumed by the corresponding forward transition t from P to P' ($P \xrightarrow{a} P'$). As will become clear with the definition of behavioral equivalences, like in [8] when going forward we view t as an *outgoing* transition of P, while when going backward we view t as an *incoming* transition of P'. The semantic rules for $\longrightarrow \subseteq \mathbb{P} \times A \times \mathbb{P}$ are defined in Table 1 and generate the labeled transition system $(\mathbb{P}, A, \longrightarrow)$ [4].

The first rule for action prefix ($\textsc{Act}_{\mathrm{f}}$ where f stands for forward) applies only if P is initial and retains the executed action in the target process of the generated forward transition by decorating the action itself with $\dagger$. The second rule for action prefix ($\textsc{Act}_{\mathrm{p}}$ where p stands for propagation) propagates actions executed by inner initial subprocesses.

In both rules for alternative composition ($\textsc{Cho}_{\mathrm{l}}$ and $\textsc{Cho}_{\mathrm{r}}$ where l stands for left and r stands for right), the subprocess that has not been selected for execution is retained as an initial subprocess in the target process of the generated transition. When both subprocesses are initial, both rules for alternative composition are applicable, otherwise only one of them can be applied and in that case it is the non-initial subprocess that can move, because the other one has been discarded at the moment of the selection.

Every state corresponding to a non-final process has at least one outgoing transition, while every state corresponding to a non-initial process has exactly one incoming transition due to the decoration of executed actions. The labeled transition system underlying an initial process turns out to be a tree, whose branching points correspond to occurrences of $+$.

Example 2.1 The labeled transition systems generated by the rules in Table 1 for the two initial processes $a\,.\,\underline{0} + a\,.\,\underline{0}$ and $a\,.\,\underline{0}$ are depicted below:

$$\begin{tikzcd}
 & a\,.\,\underline{0} + a\,.\,\underline{0} \arrow[dl, "a"'] \arrow[dr, "a"] & & a\,.\,\underline{0} \arrow[d, "a"] \\
a^{\dagger}.\,\underline{0} + a\,.\,\underline{0} & & a\,.\,\underline{0} + a^{\dagger}.\,\underline{0} & a^{\dagger}.\,\underline{0}
\end{tikzcd}$$

As far as the one on the left is concerned, we observe that, in the case of a standard process calculus, a single a-transition from $a\,.\,\underline{0} + a\,.\,\underline{0}$ to $\underline{0}$ would have been generated due to the absence of action decorations within processes. ■

2.3 Strong Forward, Reverse, and Forward-Reverse Bisimilarities

While forward bisimilarity considers only *outgoing* transitions [17, 16], reverse bisimilarity considers only *incoming* transitions. Forward-reverse bisimilarity [18] considers instead both outgoing transitions and incoming ones. Here are their *strong* versions studied in [4], where strong means not abstracting from τ-actions.

Definition 2.2 We say that $P_1, P_2 \in \mathbb{P}$ are *forward bisimilar*, written $P_1 \sim_{\rm FB} P_2$, iff $(P_1, P_2) \in \mathcal{B}$ for some forward bisimulation $\mathcal{B}$. A symmetric relation $\mathcal{B}$ over $\mathbb{P}$ is a *forward bisimulation* iff for all $(P_1, P_2) \in \mathcal{B}$ and $a \in A$:

- Whenever $P_1 \xrightarrow{a} P'_1$, then $P_2 \xrightarrow{a} P'_2$ with $(P'_1, P'_2) \in \mathcal{B}$. ■

Definition 2.3 We say that $P_1, P_2 \in \mathbb{P}$ are *reverse bisimilar*, written $P_1 \sim_{\rm RB} P_2$, iff $(P_1, P_2) \in \mathcal{B}$ for some reverse bisimulation $\mathcal{B}$. A symmetric relation $\mathcal{B}$ over $\mathbb{P}$ is a *reverse bisimulation* iff for all $(P_1, P_2) \in \mathcal{B}$ and $a \in A$:

- Whenever $P'_1 \xrightarrow{a} P_1$, then $P'_2 \xrightarrow{a} P_2$ with $(P'_1, P'_2) \in \mathcal{B}$. ■

Definition 2.4 We say that $P_1, P_2 \in \mathbb{P}$ are *forward-reverse bisimilar*, written $P_1 \sim_{\rm FRB} P_2$, iff $(P_1, P_2) \in \mathcal{B}$ for some forward-reverse bisimulation $\mathcal{B}$. A symmetric relation $\mathcal{B}$ over $\mathbb{P}$ is a *forward-reverse bisimulation* iff for all $(P_1, P_2) \in \mathcal{B}$ and $a \in A$:

- Whenever $P_1 \xrightarrow{a} P'_1$, then $P_2 \xrightarrow{a} P'_2$ with $(P'_1, P'_2) \in \mathcal{B}$.
- Whenever $P'_1 \xrightarrow{a} P_1$, then $P'_2 \xrightarrow{a} P_2$ with $(P'_1, P'_2) \in \mathcal{B}$. ■

$\sim_{\rm FRB} \subsetneq \sim_{\rm FB} \cap \sim_{\rm RB}$ with the inclusion being strict because, e.g., the two final processes $a^{\dagger} . \underline{0}$ and $a^{\dagger} . \underline{0} + c . \underline{0}$ are identified by $\sim_{\rm FB}$ (no outgoing transitions on both sides) and by $\sim_{\rm RB}$ (only an incoming a-transition on both sides), but distinguished by $\sim_{\rm FRB}$ as in the latter process action c is enabled again after undoing a (and hence there is an outgoing c-transition in addition to an outgoing a-transition). Moreover, $\sim_{\rm FB}$ and $\sim_{\rm RB}$ are incomparable because for instance:

$$a^{\dagger} . \underline{0} \sim_{\rm FB} \underline{0} \quad \text{but} \quad a^{\dagger} . \underline{0} \not\sim_{\rm RB} \underline{0}$$
$$a . \underline{0} \sim_{\rm RB} \underline{0} \quad \text{but} \quad a . \underline{0} \not\sim_{\rm FB} \underline{0}$$

Note that that $\sim_{\rm FRB} = \sim_{\rm FB}$ over initial processes, with $\sim_{\rm RB}$ strictly coarser, whilst $\sim_{\rm FRB} \neq \sim_{\rm RB}$ over final processes because, after going backward, previously discarded subprocesses come into play again in the forward direction.

Example 2.5 The two processes considered in Example 2.1 are identified by all the three equivalences. This is witnessed by any bisimulation that contains the pairs $(a . \underline{0} + a . \underline{0}, a . \underline{0})$, $(a^{\dagger} . \underline{0} + a . \underline{0}, a^{\dagger} . \underline{0})$, and $(a . \underline{0} + a^{\dagger} . \underline{0}, a^{\dagger} . \underline{0})$. ■

As observed in [4], it makes sense that $\sim_{\rm FB}$ identifies processes with a different past and that $\sim_{\rm RB}$ identifies processes with a different future, in particular with $\underline{0}$ that has neither past nor future. However, for $\sim_{\rm FB}$ this breaks compositionality with respect to alternative composition. As an example:

$$\begin{array}{lcl} a^{\dagger} . b . \underline{0} & \sim_{\rm FB} & b . \underline{0} \\ a^{\dagger} . b . \underline{0} + c . \underline{0} & \not\sim_{\rm FB} & b . \underline{0} + c . \underline{0} \end{array}$$

because in $a^{\dagger} . b . \underline{0} + c . \underline{0}$ action c is disabled due to the presence of the already executed action $a^{\dagger}$, while in $b . \underline{0} + c . \underline{0}$ action c is enabled as there are no past actions preventing it from occurring. Note that a similar phenomenon does not happen with $\sim_{\rm RB}$ as $a^{\dagger} . b . \underline{0} \not\sim_{\rm RB} b . \underline{0}$ due to the incoming a-transition of $a^{\dagger} . b . \underline{0}$.

This problem, which does not show up for $\sim_{\rm RB}$ and $\sim_{\rm FRB}$ because these two equivalences cannot identify an initial process with a non-initial one, leads to the following variant of $\sim_{\rm FB}$ that is sensitive to the presence of the past.

Definition 2.6 We say that $P_1, P_2 \in \mathbb{P}$ are *past-sensitive forward bisimilar*, written $P_1 \sim_{\mathrm{FB:ps}} P_2$, iff $(P_1, P_2) \in \mathcal{B}$ for some past-sensitive forward bisimulation $\mathcal{B}$. A relation $\mathcal{B}$ over $\mathbb{P}$ is a *past-sensitive forward bisimulation* iff it is a forward bisimulation such that $initial(P_1) \Longleftrightarrow initial(P_2)$ for all $(P_1, P_2) \in \mathcal{B}$. ■

Now $\sim_{\mathrm{FB:ps}}$ is sensitive to the presence of the past:

$$a^{\dagger} . b . \underline{0} \not\sim_{\mathrm{FB:ps}} b . \underline{0}$$

but can still identify non-initial processes having a different past:

$$a_1^{\dagger} . P \sim_{\mathrm{FB:ps}} a_2^{\dagger} . P$$

It holds that $\sim_{\mathrm{FRB}} \subsetneq \sim_{\mathrm{FB:ps}} \cap \sim_{\mathrm{RB}}$, with $\sim_{\mathrm{FRB}} = \sim_{\mathrm{FB:ps}}$ over initial processes as well as $\sim_{\mathrm{FB:ps}}$ and $\sim_{\mathrm{RB}}$ being incomparable because, e.g., for $a_1 \neq a_2$:

$$a_1^{\dagger} . P \sim_{\mathrm{FB:ps}} a_2^{\dagger} . P \quad \text{but} \quad a_1^{\dagger} . P \not\sim_{\mathrm{RB}} a_2^{\dagger} . P$$
$$a_1 . P \sim_{\mathrm{RB}} a_2 . P \quad \text{but} \quad a_1 . P \not\sim_{\mathrm{FB:ps}} a_2 . P$$

In [4] it has been shown that all the considered strong bisimilarities are congruences with respect to action prefix, while only $\sim_{\mathrm{FB:ps}}$, $\sim_{\mathrm{RB}}$, and $\sim_{\mathrm{FRB}}$ are congruences with respect to alternative composition too, with $\sim_{\mathrm{FB:ps}}$ being the coarsest congruence with respect to $+$ contained in $\sim_{\mathrm{FB}}$. Moreover, sound and complete equational characterizations have been provided for the three congruences.

2.4 Weak Forward, Reverse, and Forward-Reverse Bisimilarities

In [3] *weak* variants of forward, reverse, and forward-reverse bisimilarities have been studied, which are capable of abstracting from τ-actions. In the following definitions, $P \overset{\tau^*}{\Longrightarrow} P'$ means that $P' = P$ or there exists a nonempty sequence of finitely many τ-transitions such that the target of each of them coincides with the source of the subsequent one, with the source of the first one being P and the target of the last one being P'. Moreover, $\overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow}$ stands for an a-transition possibly preceded and followed by finitely many τ-transitions. We further let $\bar{A} = A \setminus \{\tau\}$.

Definition 2.7 We say that $P_1, P_2 \in \mathbb{P}$ are *weakly forward bisimilar*, written $P_1 \approx_{\mathrm{FB}} P_2$, iff $(P_1, P_2) \in \mathcal{B}$ for some weak forward bisimulation $\mathcal{B}$. A symmetric binary relation $\mathcal{B}$ over $\mathbb{P}$ is a *weak forward bisimulation* iff, whenever $(P_1, P_2) \in \mathcal{B}$, then:

- Whenever $P_1 \xrightarrow{\tau} P_1'$, then $P_2 \overset{\tau^*}{\Longrightarrow} P_2'$ and $(P_1', P_2') \in \mathcal{B}$.
- Whenever $P_1 \xrightarrow{a} P_1'$ for $a \in \bar{A}$, then $P_2 \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P_2'$ and $(P_1', P_2') \in \mathcal{B}$. ■

Definition 2.8 We say that $P_1, P_2 \in \mathbb{P}$ are *weakly reverse bisimilar*, written $P_1 \approx_{\mathrm{RB}} P_2$, iff $(P_1, P_2) \in \mathcal{B}$ for some weak reverse bisimulation $\mathcal{B}$. A symmetric binary relation $\mathcal{B}$ over $\mathbb{P}$ is a *weak reverse bisimulation* iff, whenever $(P_1, P_2) \in \mathcal{B}$, then:

- Whenever $P_1' \xrightarrow{\tau} P_1$, then $P_2' \overset{\tau^*}{\Longrightarrow} P_2$ and $(P_1', P_2') \in \mathcal{B}$.
- Whenever $P_1' \xrightarrow{a} P_1$ for $a \in \bar{A}$, then $P_2' \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P_2$ and $(P_1', P_2') \in \mathcal{B}$. ■

Definition 2.9 We say that $P_1, P_2 \in \mathbb{P}$ are *weakly forward-reverse bisimilar*, written $P_1 \approx_{\mathrm{FRB}} P_2$, iff $(P_1, P_2) \in \mathcal{B}$ for some weak forward-reverse bisimulation $\mathcal{B}$. A symmetric binary relation $\mathcal{B}$ over $\mathbb{P}$ is a *weak forward-reverse bisimulation* iff, whenever $(P_1, P_2) \in \mathcal{B}$, then:

- Whenever $P_1 \xrightarrow{\tau} P_1'$, then $P_2 \overset{\tau^*}{\Longrightarrow} P_2'$ and $(P_1', P_2') \in \mathcal{B}$.

- Whenever $P_1 \xrightarrow{a} P'_1$ for $a \in \bar{A}$, then $P_2 \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P'_2$ and $(P'_1, P'_2) \in \mathcal{B}$.
- Whenever $P'_1 \xrightarrow{\tau} P_1$, then $P'_2 \overset{\tau^*}{\Longrightarrow} P_2$ and $(P'_1, P'_2) \in \mathcal{B}$.
- Whenever $P'_1 \xrightarrow{a} P_1$ for $a \in \bar{A}$, then $P'_2 \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P_2$ and $(P'_1, P'_2) \in \mathcal{B}$. ■

Each of the three weak bisimilarities is strictly coarser than the corresponding strong one. Similar to the strong case, $\approx_{\mathrm{FRB}} \subsetneq \approx_{\mathrm{FB}} \cap \approx_{\mathrm{RB}}$ with $\approx_{\mathrm{FB}}$ and $\approx_{\mathrm{RB}}$ being incomparable. Unlike the strong case, $\approx_{\mathrm{FRB}} \neq \approx_{\mathrm{FB}}$ over initial processes. For instance, $\tau\,.\,a\,.\,\underline{0} + a\,.\,\underline{0} + b\,.\,\underline{0}$ and $\tau\,.\,a\,.\,\underline{0} + b\,.\,\underline{0}$ are identified by $\approx_{\mathrm{FB}}$ but told apart by $\approx_{\mathrm{FRB}}$: if the former performs a, the latter responds with τ followed by a and if it subsequently undoes a thus becoming $\tau^{\dagger}\,.\,a\,.\,\underline{0} + b\,.\,\underline{0}$ in which only a is enabled, the latter can only respond by undoing a thus becoming $\tau\,.\,a\,.\,\underline{0} + a\,.\,\underline{0} + b\,.\,\underline{0}$ in which both a and b are enabled. An analogous counterexample with non-initial τ-actions is given by $c\,.\,(\tau\,.\,a\,.\,\underline{0} + a\,.\,\underline{0} + b\,.\,\underline{0})$ and $c\,.\,(\tau\,.\,a\,.\,\underline{0} + b\,.\,\underline{0})$.

As observed in [3], $\approx_{\mathrm{FB}}$ suffers from the same compositionality problem with respect to alternative composition as $\sim_{\mathrm{FB}}$. Moreover, $\approx_{\mathrm{FB}}$ and $\approx_{\mathrm{FRB}}$ feature the same compositionality problem as weak bisimilarity for standard forward-only processes [16], i.e., for $\approx \in \{\approx_{\mathrm{FB}}, \approx_{\mathrm{FRB}}\}$ it holds that:

$$\begin{array}{rcl} \tau\,.\,a\,.\,\underline{0} & \approx & a\,.\,\underline{0} \\ \tau\,.\,a\,.\,\underline{0} + b\,.\,\underline{0} & \not\approx & a\,.\,\underline{0} + b\,.\,\underline{0} \end{array}$$

because if $\tau\,.\,a\,.\,\underline{0} + b\,.\,\underline{0}$ performs τ thereby evolving to $\tau^{\dagger}\,.\,a\,.\,\underline{0} + b\,.\,\underline{0}$ where only a is enabled in the forward direction, then $a\,.\,\underline{0} + b\,.\,\underline{0}$ can neither move nor idle in the attempt to evolve in such a way to match $\tau^{\dagger}\,.\,a\,.\,\underline{0} + b\,.\,\underline{0}$.

To solve both problems it is sufficient to redefine the two equivalences by making them sensitive to the presence of the past, exactly as in the strong case for forward bisimilarity. By so doing, $\tau\,.\,a\,.\,\underline{0}$ is no longer identified with $a\,.\,\underline{0}$: if the former performs τ thereby evolving to $\tau^{\dagger}\,.\,a\,.\,\underline{0}$ and the latter idles, then $\tau^{\dagger}\,.\,a\,.\,\underline{0}$ and $a\,.\,\underline{0}$ are told apart because they are not both initial or non-initial.

Definition 2.10 We say that $P_1, P_2 \in \mathbb{P}$ are *weakly past-sensitive forward bisimilar*, written $P_1 \approx_{\mathrm{FB:ps}} P_2$, iff $(P_1, P_2) \in \mathcal{B}$ for some weak past-sensitive forward bisimulation $\mathcal{B}$. A binary relation $\mathcal{B}$ over $\mathbb{P}$ is a *weak past-sensitive forward bisimulation* iff it is a weak forward bisimulation such that $\mathit{initial}(P_1) \Longleftrightarrow \mathit{initial}(P_2)$ for all $(P_1, P_2) \in \mathcal{B}$. ■

Definition 2.11 We say that $P_1, P_2 \in \mathbb{P}$ are *weakly past-sensitive forward-reverse bisimilar*, written $P_1 \approx_{\mathrm{FRB:ps}} P_2$, iff $(P_1, P_2) \in \mathcal{B}$ for some weak past-sensitive forward-reverse bisimulation $\mathcal{B}$. A binary relation $\mathcal{B}$ over $\mathbb{P}$ is a *weak past-sensitive forward-reverse bisimulation* iff it is a weak forward-reverse bisimulation such that $\mathit{initial}(P_1) \Longleftrightarrow \mathit{initial}(P_2)$ for all $(P_1, P_2) \in \mathcal{B}$. ■

Like in the non-past-sensitive case, $\approx_{\mathrm{FRB:ps}} \neq \approx_{\mathrm{FB:ps}}$ over initial processes, as shown by $\tau\,.\,a\,.\,\underline{0} + a\,.\,\underline{0}$ and $\tau\,.\,a\,.\,\underline{0}$: if the former performs a, the latter responds with τ followed by a and if it subsequently undoes a thus becoming the non-initial process $\tau^{\dagger}\,.\,a\,.\,\underline{0}$, the latter can only respond by undoing a thus becoming the initial process $\tau\,.\,a\,.\,\underline{0} + a\,.\,\underline{0}$. An analogous counterexample with non-initial τ-actions is given again by $c\,.\,(\tau\,.\,a\,.\,\underline{0} + a\,.\,\underline{0} + b\,.\,\underline{0})$ and $c\,.\,(\tau\,.\,a\,.\,\underline{0} + b\,.\,\underline{0})$.

Observing that $\sim_{\mathrm{FRB}} \subsetneq \approx_{\mathrm{FRB:ps}}$ as the former naturally satisfies the initiality condition, in [3] it has been shown that all the considered weak bisimilarities are congruences with respect to action prefix, while only $\approx_{\mathrm{FB:ps}}$, $\approx_{\mathrm{RB}}$, and $\approx_{\mathrm{FRB:ps}}$ are congruences with respect to alternative composition too, with $\approx_{\mathrm{FB:ps}}$ and $\approx_{\mathrm{FRB:ps}}$ respectively being the coarsest congruences with respect to $+$ contained in $\approx_{\mathrm{FB}}$ and $\approx_{\mathrm{FRB}}$. Sound and complete equational characterizations have been provided for the three congruences.

3 Modal Logic Characterizations

In this section we investigate modal logic characterizations for the three strong bisimilarities $\sim_{\mathrm{FB}}$, $\sim_{\mathrm{RB}}$, and $\sim_{\mathrm{FRB}}$, the three weak bisimilarities $\approx_{\mathrm{FB}}$, $\approx_{\mathrm{RB}}$, and $\approx_{\mathrm{FRB}}$, and the three past-sensitive variants $\sim_{\mathrm{FB:ps}}$, $\approx_{\mathrm{FB:ps}}$, and $\approx_{\mathrm{FRB:ps}}$.

We start by introducing a general modal logic $\mathscr{L}$ from which we will take nine fragments to characterize the nine aforementioned bisimilarities. It consists of Hennessy-Milner logic [11] extended with the proposition init, the strong backward modality $\langle a^\dagger \rangle$, the two weak forward modalities $\langle\langle \tau \rangle\rangle$ and $\langle\langle a \rangle\rangle$, and the two weak backward modalities $\langle\langle \tau^\dagger \rangle\rangle$ and $\langle\langle a^\dagger \rangle\rangle$ (where $a \in \bar{A}$ within weak modalities):

$$\phi \ ::= \ \mathrm{true} \mid \mathrm{init} \mid \neg\phi \mid \phi \wedge \phi \mid \langle a \rangle \phi \mid \langle a^\dagger \rangle \phi \mid \langle\langle \tau \rangle\rangle \phi \mid \langle\langle a \rangle\rangle \phi \mid \langle\langle \tau^\dagger \rangle\rangle \phi \mid \langle\langle a^\dagger \rangle\rangle \phi$$

The satisfaction relation $\models \subseteq \mathbb{P} \times \mathscr{L}$ is defined by induction on the syntactical structure of the formulas as follows:

$$\begin{array}{lll}
P \models \mathrm{true} & & \text{for all } P \in \mathbb{P} \\
P \models \mathrm{init} & & \text{iff } \mathit{initial}(P) \\
P \models \neg\phi & & \text{iff } P \not\models \phi \\
P \models \phi_1 \wedge \phi_2 & & \text{iff } P \models \phi_1 \text{ and } P \models \phi_2 \\
P \models \langle a \rangle \phi & & \text{iff there exists } P' \in \mathbb{P} \text{ such that } P \xrightarrow{a} P' \text{ and } P' \models \phi \\
P \models \langle a^\dagger \rangle \phi & & \text{iff there exists } P' \in \mathbb{P} \text{ such that } P' \xrightarrow{a} P \text{ and } P' \models \phi \\
P \models \langle\langle \tau \rangle\rangle \phi & & \text{iff there exists } P' \in \mathbb{P} \text{ such that } P \overset{\tau^*}{\Longrightarrow} P' \text{ and } P' \models \phi \\
P \models \langle\langle a \rangle\rangle \phi & & \text{iff there exists } P' \in \mathbb{P} \text{ such that } P \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P' \text{ and } P' \models \phi \\
P \models \langle\langle \tau^\dagger \rangle\rangle \phi & & \text{iff there exists } P' \in \mathbb{P} \text{ such that } P' \overset{\tau^*}{\Longrightarrow} P \text{ and } P' \models \phi \\
P \models \langle\langle a^\dagger \rangle\rangle \phi & & \text{iff there exists } P' \in \mathbb{P} \text{ such that } P' \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P \text{ and } P' \models \phi
\end{array}$$

The use of backward operators is not new in the definition of properties of programs through temporal logics [15] or modal logics [12]. In particular, in the latter work a logic with a past operator was introduced to capture interesting properties of generalized labeled transition systems where only visible actions are considered, in which setting it is proved that the equivalence induced by the considered logic coincides with a generalization of the standard forward-only strong bisimilarity of [16]. This result was later confirmed in [9] where it is shown that the addition of a strong backward modality (interpreted over computation paths instead of states) provides no additional discriminating power with respect to the Hennessy-Milner logic, i.e., the induced equivalence is again strong bisimilarity.

In contrast, in our context – in which all equivalences are defined over states – the strong forward bisimilarities $\sim_{\mathrm{FB}}$ and $\sim_{\mathrm{FB:ps}}$ do not coincide with the strong forward-reverse bisimilarity $\sim_{\mathrm{FRB}}$ and this extends to their weak counterparts. In other words, the presence of backward modalities matters. It is worth noting that our two weak backward modalities are similar to the ones considered in [8, 9] to characterize weak back-and-forth bisimilarity (defined over computation paths), which is finer than the standard forward-only weak bisimilarity of [16] and coincides with branching bisimilarity [10].

By taking suitable fragments of $\mathscr{L}$ we can characterize all the nine bisimilarities introduced in Section 2. For each of the four strong bisimilarities $\sim_B$, where $B \in \{\mathrm{FB}, \mathrm{FB{:}ps}, \mathrm{RB}, \mathrm{FRB}\}$, we can define the corresponding logic $\mathscr{L}_B$. The same can be done for each of the five weak bisimilarities $\approx_B$, where $B \in \{\mathrm{FB}, \mathrm{FB{:}ps}, \mathrm{RB}, \mathrm{FRB}, \mathrm{FRB{:}ps}\}$, to obtain the corresponding logic $\mathscr{L}_B^\tau$. All the considered fragments can be found in Table 2, which indicates that the proposition init is needed only for the past-sensitive bisimilarities. The forthcoming Theorems 3.1 and 3.2 show that each such fragment induces the intended bisimilarity, in the sense that two processes are bisimilar iff they satisfy the same set of formulas of the fragment at hand.

	true	init	$\neg$	$\wedge$	$\langle a \rangle$	$\langle a^\dagger \rangle$	$\langle\langle \tau \rangle\rangle$	$\langle\langle a \rangle\rangle$	$\langle\langle \tau^\dagger \rangle\rangle$	$\langle\langle a^\dagger \rangle\rangle$
$\mathscr{L}_{\mathrm{FB}}$	✓		✓	✓	✓					
$\mathscr{L}_{\mathrm{FB:ps}}$	✓	✓	✓	✓	✓					
$\mathscr{L}_{\mathrm{RB}}$	✓					✓				
$\mathscr{L}_{\mathrm{FRB}}$	✓		✓	✓	✓	✓				
$\mathscr{L}^{\tau}_{\mathrm{FB}}$	✓		✓	✓			✓	✓		
$\mathscr{L}^{\tau}_{\mathrm{FB:ps}}$	✓	✓	✓	✓			✓	✓		
$\mathscr{L}^{\tau}_{\mathrm{RB}}$	✓								✓	✓
$\mathscr{L}^{\tau}_{\mathrm{FRB}}$	✓		✓	✓			✓	✓	✓	✓
$\mathscr{L}^{\tau}_{\mathrm{FRB:ps}}$	✓	✓	✓	✓			✓	✓	✓	✓

Table 2: Fragments of $\mathscr{L}$ characterizing the considered bisimilarities

The technique used to prove the two theorems is inspired by the one employed in [1] to show that Hennessy-Milner logic characterizes the strong forward-only bisimilarity of [16]. The two implications of either theorem are demonstrated separately. To prove that any pair of bisimilar processes P_1 and P_2 satisfy the same formulas of the considered fragment, we assume that $P_1 \models \phi$ for some formula ϕ and then we proceed by induction on the depth of ϕ to show that $P_2 \models \phi$ too, where the depth of a formula is defined by induction on the syntactical structure of the formula itself as follows:

$$\begin{array}{rcl}
depth(\mathrm{true}) & = & 1 \\
depth(\mathrm{init}) & = & 1 \\
depth(\neg\phi) & = & 1 + depth(\phi) \\
depth(\phi_1 \wedge \phi_2) & = & 1 + \max(depth(\phi_1), depth(\phi_2)) \\
depth(\langle a \rangle \phi) & = & 1 + depth(\phi) \\
depth(\langle a^\dagger \rangle \phi) & = & 1 + depth(\phi) \\
depth(\langle\langle \tau \rangle\rangle \phi) & = & 1 + depth(\phi) \\
depth(\langle\langle a \rangle\rangle \phi) & = & 1 + depth(\phi) \\
depth(\langle\langle \tau^\dagger \rangle\rangle \phi) & = & 1 + depth(\phi) \\
depth(\langle\langle a^\dagger \rangle\rangle \phi) & = & 1 + depth(\phi)
\end{array}$$

As for the reverse implication, we show that the relation $\mathscr{B}$ formed by all pairs of processes (P_1, P_2) that satisfy the same formulas of the considered fragment is a bisimulation. More specifically, starting from $(P_1, P_2) \in \mathscr{B}$ we proceed by contradiction by assuming that, whenever P_1 has a move to/from P'_1 with an action a, then there is no P'_2 such that P_2 has a move to/from P'_2 with a and $(P'_1, P'_2) \in \mathscr{B}$. This entails that, for every P_{2_i} forward/backward reachable from P_2 by performing a, by definition of $\mathscr{B}$ there exists some formula ϕ_i such that $P'_1 \models \phi_i$ and $P'_{2_i} \not\models \phi_i$, which leads to a formula with a forward/backward modality on a followed by $\bigwedge_i \phi_i$ that is satisfied by P_1 but not by P_2, thereby contradicting $(P_1, P_2) \in \mathscr{B}$.

Theorem 3.1 Let $P_1, P_2 \in \mathbb{P}$ and $B \in \{\mathrm{FB}, \mathrm{FB{:}ps}, \mathrm{RB}, \mathrm{FRB}\}$. Then $P_1 \sim_B P_2 \iff \forall \phi \in \mathscr{L}_B . P_1 \models \phi \Leftrightarrow P_2 \models \phi$. ■

Theorem 3.2 Let $P_1, P_2 \in \mathbb{P}$ and $B \in \{\mathrm{FB}, \mathrm{FB{:}ps}, \mathrm{RB}, \mathrm{FRB}, \mathrm{FRB{:}ps}\}$. Then $P_1 \approx_B P_2 \iff \forall \phi \in \mathscr{L}^{\tau}_B$. $P_1 \models \phi \Leftrightarrow P_2 \models \phi$. ■

We conclude with the following observations:

- The fragments that characterize the four forward bisimilarities $\sim_{\mathrm{FB}}$, $\sim_{\mathrm{FB:ps}}$, $\approx_{\mathrm{FB}}$, and $\approx_{\mathrm{FB:ps}}$ are essentially identical to the Hennessy-Milner logic (first two bisimilarities) and its weak variant (last two bisimilarities). The only difference is the possible presence of the proposition init, which is needed to distinguish between initial and non-initial processes in the past-sensitive cases.

- The fragments that characterize the two reverse bisimilarities $\sim_{\mathrm{RB}}$ and $\approx_{\mathrm{RB}}$ only include true and the backward modalities $\langle a^{\dagger}\rangle$ (first bisimilarity) and $\langle\langle \tau^{\dagger}\rangle\rangle$ and $\langle\langle a^{\dagger}\rangle\rangle$ (second bisimilarity). The absence of conjunction reflects the fact that, when going backward, processes must follow exactly the sequence of actions they performed in the forward direction and hence no choice is involved, consistent with every non-initial process having precisely one incoming transition. In other words, the strong and weak reverse bisimilarities boil down to strong and weak reverse trace equivalences, respectively, which consider traces obtained when going in the backward direction.
- The fragments that characterize the three forward-reverse bisimilarities $\sim_{\mathrm{FRB}}$, $\approx_{\mathrm{FRB}}$, and $\approx_{\mathrm{FRB:PS}}$ are akin to the logic $\mathscr{L}_{\mathrm{BF}}$ introduced in [8] to characterize weak back-and-forth bisimilarity and branching bisimilarity. A crucial distinction between our three fragments and $\mathscr{L}_{\mathrm{BF}}$ is that the former are interpreted over states while $\mathscr{L}_{\mathrm{BF}}$ is interpreted over computation paths. Moreover, as already mentioned, defining a strong variant of $\mathscr{L}_{\mathrm{BF}}$ would yield a logic that characterizes strong bisimilarity, whereas in our setting forward-only bisimilarities are different from forward-reverse ones and hence different logics are needed.

4 Weak Forward-Reverse Bisimilarity and Branching Bisimilarity

In this section we establish a clear connection between weak forward-reverse bisimilarity and branching bisimilarity [10]. Unlike the standard forward-only weak bisimilarity of [16], branching bisimilarity preserves the branching structure of processes even when abstracting from τ-actions.

Definition 4.1 We say that $P_1, P_2 \in \mathbb{P}$ are *branching bisimilar*, written $P_1 \approx_{\mathrm{BB}} P_2$, iff $(P_1, P_2) \in \mathscr{B}$ for some branching bisimulation $\mathscr{B}$. A symmetric binary relation $\mathscr{B}$ over $\mathbb{P}$ is a *branching bisimulation* iff, whenever $(P_1, P_2) \in \mathscr{B}$, then for all $P_1 \xrightarrow{a} P_1'$ it holds that:

- either $a = \tau$ and $(P_1', P_2) \in \mathscr{B}$;
- or $P_2 \overset{\tau^*}{\Longrightarrow} \bar{P}_2 \xrightarrow{a} P_2'$ with $(P_1, \bar{P}_2) \in \mathscr{B}$ and $(P_1', P_2') \in \mathscr{B}$. ■

Branching bisimilarity is known to have some relationships with reversibility. More precisely, in [8] strong and weak back-and-forth bisimilarities have been introduced over labeled transition systems – where outgoing transitions are considered in the forward bisimulation game while incoming transitions are considered in the backward bisimulation game – and respectively shown to coincide with the standard forward-only strong bisimilarity of [16] and branching bisimilarity.

In the setting of [8], strong and weak back-and-forth bisimilarities have been defined over computation paths rather than states so that, in the presence of concurrency, any backward computation is *constrained* to follow the same path as the corresponding forward computation, which is consistent with an interleaving view of parallel composition. This is quite different from the forward-reverse bisimilarity over states defined in [18], which accounts for the fact that when going backward the order in which independent transitions are undone may be different from the order in which they were executed in the forward direction, thus leading to a truly concurrent semantics.

Since in our setting we consider only sequential processes, hence any backward computation *naturally* follows the same path as the corresponding forward computation, we are neutral with respect to interleaving vs. true concurrency. Like in [8] we define a single transition relation and then we distinguish between outgoing transitions and incoming transitions in the bisimulation game. However, unlike [8], our bisimilarities are defined over states as in [16, 10, 18], not over paths. In the rest of this section we show that our weak forward-reverse bisimilarity *over states* coincides with branching bisimilarity by following the proof strategy adopted in [8] for weak back-and-forth bisimilarity.

First of all, we prove that, like branching bisimilarity, our weak forward-reverse bisimilarity satisfies the *stuttering property* [10]. This means that, given a sequence of finitely many τ-transitions, if the source process of the first transition and the target process of the last transition are equivalent to each other, then all the intermediate processes are equivalent to them too – see $P_2 \overset{\tau^*}{\Longrightarrow} \bar{P}_2$ in Definition 4.1 when $P_1, P_2, \bar{P}_2$ are pairwise related by the maximal branching bisimulation $\approx_{\mathrm{BB}}$. In other words, while traversing the considered sequence of τ-transitions, we remain in the same equivalence class of processes, not only in the forward direction but – as we are talking about weak forward-reverse bisimilarity – *also in the backward direction*. This property does not hold in the case of the standard forward-only weak bisimilarity of [16].

Lemma 4.2 Let $n \in \mathbb{N}_{>0}$, $P_i \in \mathbb{P}$ for all $0 \leq i \leq n$, and $P_i \xrightarrow{\tau} P_{i+1}$ for all $0 \leq i \leq n-1$. If $P_0 \approx_{\mathrm{FRB}} P_n$ then $P_i \approx_{\mathrm{FRB}} P_0$ for all $0 \leq i \leq n$.

Proof Consider the reflexive and symmetric binary relation $\mathcal{B} = \cup_{i \in \mathbb{N}} \mathcal{B}_i$ over $\mathbb{P}$ where:

- $\mathcal{B}_0 = \approx_{\mathrm{FRB}}$.
- $\mathcal{B}_i = \mathcal{B}_{i-1} \cup \{(P,P'), (P',P) \in \mathbb{P} \times \mathbb{P} \mid \exists P'' \in \mathbb{P}.\, (P,P'') \in \mathcal{B}_{i-1} \wedge P \overset{\tau^*}{\Longrightarrow} P' \xrightarrow{\tau} P''\}$ for all $i \in \mathbb{N}_{>0}$.

We start by proving that $\mathcal{B}$ satisfies the stuttering property, i.e., given $n \in \mathbb{N}_{>0}$ and $P_i \in \mathbb{P}$ for all $0 \leq i \leq n$, if $P_i \xrightarrow{\tau} P_{i+1}$ for all $0 \leq i \leq n-1$ and $(P_0, P_n) \in \mathcal{B}$, then $(P_i, P_0) \in \mathcal{B}$ for all $0 \leq i \leq n$. We proceed by induction on n:

- If $n = 1$ then the considered computation is simply $P_0 \xrightarrow{\tau} P_1$ with $(P_0, P_1) \in \mathcal{B}$ and hence trivially $(P_i, P_0) \in \mathcal{B}$ for all $0 \leq i \leq 1$ as $\mathcal{B}$ is reflexive – $(P_0, P_0) \in \mathcal{B}$ – and symmetric – $(P_1, P_0) \in \mathcal{B}$.
- Let $n > 1$. Since $(P_0, P_n) \in \mathcal{B}$, there must exist $m \in \mathbb{N}$ such that $(P_0, P_n) \in \mathcal{B}_m$. Let us consider the smallest such m. Then $(P_0, P_{n-1}) \in \mathcal{B}_{m+1}$ by definition of $\mathcal{B}_{m+1}$, hence $(P_0, P_{n-1}) \in \mathcal{B}$. From the induction hypothesis it follows that $(P_i, P_0) \in \mathcal{B}$ for all $0 \leq i \leq n-1$, hence $(P_i, P_0) \in \mathcal{B}$ for all $0 \leq i \leq n$ because $(P_0, P_n) \in \mathcal{B}$ and $\mathcal{B}$ is symmetric so that $(P_n, P_0) \in \mathcal{B}$.

We now prove that every symmetric relation $\mathcal{B}_i$ is a weak forward-reverse bisimulation. We proceed by induction on $i \in \mathbb{N}$:

- If $i = 0$ then $\mathcal{B}_i$ is the maximal weak forward-reverse bisimulation.
- Let $i \geq 1$ and suppose that $\mathcal{B}_{i-1}$ is a weak forward-reverse bisimulation. Given $(P, P') \in \mathcal{B}_i$, assume that $P \xrightarrow{a} Q$ (resp. $Q \xrightarrow{a} P$) where $a \in A$. There are two cases:
  - If $(P, P') \in \mathcal{B}_{i-1}$ then by the induction hypothesis $a = \tau$ and $P' \overset{\tau^*}{\Longrightarrow} Q'$ (resp. $Q' \overset{\tau^*}{\Longrightarrow} P'$) or $a \neq \tau$ and $P' \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} Q'$ (resp. $Q' \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P'$) with $(Q, Q') \in \mathcal{B}_{i-1}$ and hence $(Q, Q') \in \mathcal{B}_i$ as $\mathcal{B}_{i-1} \subseteq \mathcal{B}_i$ by definition of $\mathcal{B}_i$.
  - If instead $(P, P') \notin \mathcal{B}_{i-1}$ then from $(P, P') \in \mathcal{B}_i$ it follows that $\exists P'' \in \mathbb{P}.\, (P, P'') \in \mathcal{B}_{i-1} \wedge P \overset{\tau^*}{\Longrightarrow} P' \xrightarrow{\tau} P''$. There are two subcases:
    - $*$ In the forward case, i.e., $P \xrightarrow{a} Q$, there are two further subcases:

- If $(Q,P'') \in \mathscr{B}_{i-1}$ and $a = \tau$, then from $P' \xrightarrow{\tau} P''$ it follows that $P' \overset{\tau^*}{\Longrightarrow} P''$ with $(Q,P'') \in \mathscr{B}_i$ as $\mathscr{B}_{i-1} \subseteq \mathscr{B}_i$.
- Otherwise from $(P,P'') \in \mathscr{B}_{i-1}$ and the induction hypothesis it follows that $P'' \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P'''$ with $(Q,P''') \in \mathscr{B}_{i-1}$ so that $P' \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P'''$ with $(Q,P''') \in \mathscr{B}_i$ as $\mathscr{B}_{i-1} \subseteq \mathscr{B}_i$.

$*$ In the backward case, i.e., $Q \xrightarrow{a} P$, it suffices to note that from $P \overset{\tau^*}{\Longrightarrow} P'$ it follows that $Q \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P'$.

Since $\mathscr{B}$ is the union of countably many weak forward-reverse bisimulations, it holds that $\mathscr{B} \subseteq \approx_{\text{FRB}}$. On the other hand, $\approx_{\text{FRB}} \subseteq \mathscr{B}$ by definition of $\mathscr{B}_0$. In conclusion $\mathscr{B} = \approx_{\text{FRB}}$ – i.e., no relation $\mathscr{B}_i$ for $i \in \mathbb{N}_{>0}$ adds further pairs with respect to $\mathscr{B}_0$ – and hence $\approx_{\text{FRB}}$ satisfies the stuttering property because so does $\mathscr{B}$. ■

Note that the lemma above considers $\approx_{\text{FRB}}$, not $\approx_{\text{FRB:PS}}$. Indeed the stuttering property does not hold for $\approx_{\text{FRB:PS}}$ when *initial*(P_0), because in that case a τ-action would be decorated inside P_1 and hence $P_1 \not\approx_{\text{FRB:ps}} P_0$. Therefore $\approx_{\text{FRB:PS}}$ satisfies the stuttering property only over non-initial processes.

Secondly, we prove that $\approx_{\text{FRB}}$ satisfies the *cross property* [8]. This means that, whenever two processes reachable from two $\approx_{\text{FRB}}$-equivalent processes can perform a sequence of finitely many τ-transitions such that each of the two target processes is $\approx_{\text{FRB}}$-equivalent to the source process of the other sequence, then the two target processes are $\approx_{\text{FRB}}$-equivalent to each other as well.

Lemma 4.3 Let $P_1, P_2 \in \mathbb{P}$ be such that $P_1 \approx_{\text{FRB}} P_2$. For all $P_1', P_1'' \in \mathbb{P}$ reachable from P_1 such that $P_1' \overset{\tau^*}{\Longrightarrow} P_1''$ and for all $P_2', P_2'' \in \mathbb{P}$ reachable from P_2 such that $P_2' \overset{\tau^*}{\Longrightarrow} P_2''$, if $P_1' \approx_{\text{FRB}} P_2''$ and $P_1'' \approx_{\text{FRB}} P_2'$ then $P_1'' \approx_{\text{FRB}} P_2''$.

Proof Given $P_1, P_2 \in \mathbb{P}$ with $P_1 \approx_{\text{FRB}} P_2$, consider the symmetric relation $\mathscr{B} = \approx_{\text{FRB}} \cup \{(P_1'', P_2''), (P_2'', P_1'') \in \mathbb{P} \times \mathbb{P} \mid \exists P_1', P_2' \in \mathbb{P}$ resp. reachable from $P_1, P_2.\ P_1' \overset{\tau^*}{\Longrightarrow} P_1'' \wedge P_2' \overset{\tau^*}{\Longrightarrow} P_2'' \wedge P_1' \approx_{\text{FRB}} P_2'' \wedge P_1'' \approx_{\text{FRB}} P_2'\}$. The result follows by proving that $\mathscr{B}$ is a weak forward-reverse bisimulation, because this implies that $P_1'' \approx_{\text{FRB}} P_2''$ for every additional pair – i.e., $\mathscr{B}$ satisfies the cross property – as well as $\mathscr{B} = \approx_{\text{FRB}}$ – hence $\approx_{\text{FRB}}$ satisfies the cross property too.
Let $(P_1'', P_2'') \in \mathscr{B} \setminus \approx_{\text{FRB}}$ to avoid trivial cases. Then there exist $P_1', P_2' \in \mathbb{P}$ respectively reachable from P_1, P_2 such that $P_1' \overset{\tau^*}{\Longrightarrow} P_1''$, $P_2' \overset{\tau^*}{\Longrightarrow} P_2''$, $P_1' \approx_{\text{FRB}} P_2''$, and $P_1'' \approx_{\text{FRB}} P_2'$. There are two cases:

- In the forward case, assume that $P_1'' \xrightarrow{a} P_1'''$, from which it follows that $P_1' \overset{\tau^*}{\Longrightarrow} P_1'' \xrightarrow{a} P_1'''$. Since $P_1' \approx_{\text{FRB}} P_2''$, we obtain $P_2'' \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P_2'''$, or $P_2'' \overset{\tau^*}{\Longrightarrow} P_2'''$ when $a = \tau$, with $P_1''' \approx_{\text{FRB}} P_2'''$ and hence $(P_1''', P_2''') \in \mathscr{B}$. Starting from $P_2'' \xrightarrow{a} P_2'''$ one exploits $P_2' \overset{\tau^*}{\Longrightarrow} P_2''$ and $P_1'' \approx_{\text{FRB}} P_2'$ instead.
- In the backward case, assume that $P_1''' \xrightarrow{a} P_1''$. Since $P_1'' \approx_{\text{FRB}} P_2'$, we obtain $P_2''' \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P_2'$, so that $P_2''' \overset{\tau^*}{\Longrightarrow} \xrightarrow{a} \overset{\tau^*}{\Longrightarrow} P_2''$, or $P_2''' \overset{\tau^*}{\Longrightarrow} P_2'$ when $a = \tau$, so that $P_2''' \overset{\tau^*}{\Longrightarrow} P_2''$, with $P_1''' \approx_{\text{FRB}} P_2'''$ and hence $(P_1''', P_2''') \in \mathscr{B}$. Starting from $P_2''' \xrightarrow{a} P_2''$ one exploits $P_1' \approx_{\text{FRB}} P_2''$ and $P_1' \overset{\tau^*}{\Longrightarrow} P_1''$ instead. ■

We are now in a position of proving that $\approx_{\text{FRB}}$ coincides with $\approx_{\text{BB}}$. This only holds over initial processes though. As an example, $a_1^{\dagger}.\,b\,.\,P \approx_{\text{BB}} a_2^{\dagger}.\,b\,.\,P$ but $a_1^{\dagger}.\,b\,.\,P \not\approx_{\text{FRB}} a_2^{\dagger}.\,b\,.\,P$ when $a_1 \neq a_2$.

Theorem 4.4 Let $P_1, P_2 \in \mathbb{P}$ be initial. Then $P_1 \approx_{\text{FRB}} P_2$ iff $P_1 \approx_{\text{BB}} P_2$.

Proof Given two initial processes $P_1, P_2 \in \mathbb{P}$, we divide the proof into two parts:

- Given a weak forward-reverse bisimulation $\mathcal{B}$ witnessing $P_1 \approx_{\rm FRB} P_2$ and only containing all the pairs of $\approx_{\rm FRB}$-equivalent processes reachable from P_1 and P_2 so that Lemma 4.3 is applicable to $\mathcal{B}$, we prove that $\mathcal{B}$ is a branching bisimulation too. Let $(Q_1, Q_2) \in \mathcal{B}$, where Q_1 is reachable from P_1 while Q_2 is reachable from P_2, and assume that $Q_1 \xrightarrow{a} Q'_1$. There are two cases:
  - Suppose that $a = \tau$ and $Q_2 \stackrel{\tau^*}{\Longrightarrow} Q'_2$ with $(Q'_1, Q'_2) \in \mathcal{B}$. This means that we have a sequence of $n \geq 0$ transitions of the form $Q_{2,i} \xrightarrow{\tau} Q_{2,i+1}$ for all $0 \leq i \leq n-1$ where $Q_{2,0}$ is Q_2 while $Q_{2,n}$ is Q'_2 so that $(Q'_1, Q_{2,n}) \in \mathcal{B}$.
    If $n = 0$ then Q'_2 is Q_2 and we are done because $(Q'_1, Q_2) \in \mathcal{B}$, otherwise from $Q_{2,n}$ we go back to $Q_{2,n-1}$ via $Q_{2,n-1} \xrightarrow{\tau} Q_{2,n}$. If Q'_1 stays idle so that $(Q'_1, Q_{2,n-1}) \in \mathcal{B}$ and $n = 1$ then we are done because $(Q'_1, Q_2) \in \mathcal{B}$, otherwise we go back to $Q_{2,n-2}$ via $Q_{2,n-2} \xrightarrow{\tau} Q_{2,n-1}$. By repeating this procedure, either we get to $(Q'_1, Q_{2,0}) \in \mathcal{B}$ and we are done because $(Q'_1, Q_2) \in \mathcal{B}$, or for some $0 < m \leq n$ such that $(Q'_1, Q_{2,m}) \in \mathcal{B}$ we have that the incoming transition $Q_{2,m-1} \xrightarrow{\tau} Q_{2,m}$ is matched by $\bar{Q}_1 \stackrel{\tau^*}{\Longrightarrow} Q_1 \xrightarrow{\tau} Q'_1$ with $(\bar{Q}_1, Q_{2,m-1}) \in \mathcal{B}$. In the latter case, since $\bar{Q}_1 \stackrel{\tau^*}{\Longrightarrow} Q_1$, $Q_2 \stackrel{\tau^*}{\Longrightarrow} Q_{2,m-1}$, $(\bar{Q}_1, Q_{2,m-1}) \in \mathcal{B}$, and $(Q_1, Q_2) \in \mathcal{B}$, from Lemma 4.3 it follows that $(Q_1, Q_{2,m-1}) \in \mathcal{B}$. In conclusion $Q_2 \stackrel{\tau^*}{\Longrightarrow} Q_{2,m-1} \xrightarrow{\tau} Q_{2,m}$ with $(Q_1, Q_{2,m-1}) \in \mathcal{B}$ and $(Q'_1, Q_{2,m}) \in \mathcal{B}$.
  - Suppose that $a \neq \tau$ and $Q_2 \stackrel{\tau^*}{\Longrightarrow} \bar{Q}_2 \xrightarrow{a} \bar{Q}'_2 \stackrel{\tau^*}{\Longrightarrow} Q'_2$ with $(Q'_1, Q'_2) \in \mathcal{B}$.
    From $\bar{Q}'_2 \stackrel{\tau^*}{\Longrightarrow} Q'_2$ and $(Q'_1, Q'_2) \in \mathcal{B}$ it follows that $\bar{Q}'_1 \stackrel{\tau^*}{\Longrightarrow} Q'_1$ with $(\bar{Q}'_1, \bar{Q}'_2) \in \mathcal{B}$. Since Q'_1 already has an incoming a-transition from Q_1 and every non-initial process has exactly one incoming transition, we derive that $\bar{Q}'_1$ is Q'_1 and hence $(Q'_1, \bar{Q}'_2) \in \mathcal{B}$.
    From $\bar{Q}_2 \xrightarrow{a} \bar{Q}'_2$ and $(Q'_1, \bar{Q}'_2) \in \mathcal{B}$ it follows that $\bar{Q}_1 \stackrel{\tau^*}{\Longrightarrow} Q_1 \xrightarrow{a} Q'_1$ with $(\bar{Q}_1, \bar{Q}_2) \in \mathcal{B}$. Since $\bar{Q}_1 \stackrel{\tau^*}{\Longrightarrow} Q_1$, $Q_2 \stackrel{\tau^*}{\Longrightarrow} \bar{Q}_2$, $(\bar{Q}_1, \bar{Q}_2) \in \mathcal{B}$, and $(Q_1, Q_2) \in \mathcal{B}$, from Lemma 4.3 it follows that $(Q_1, \bar{Q}_2) \in \mathcal{B}$.
    In conclusion $Q_2 \stackrel{\tau^*}{\Longrightarrow} \bar{Q}_2 \xrightarrow{a} \bar{Q}'_2$ with $(Q_1, \bar{Q}_2) \in \mathcal{B}$ and $(Q'_1, \bar{Q}'_2) \in \mathcal{B}$.
- Given a branching bisimulation $\mathcal{B}$ witnessing $P_1 \approx_{\rm BB} P_2$ and only containing all the processes reachable from P_1 and P_2, we prove that $\mathcal{B}$ is a weak forward-reverse bisimulation too. Let $(Q_1, Q_2) \in \mathcal{B}$ with Q_1 reachable from P_1 and Q_2 reachable from P_2. There are two cases:
  - In the forward case, assume that $Q_1 \xrightarrow{a} Q'_1$. Then either $a = \tau$ and $(Q'_1, Q_2) \in \mathcal{B}$, hence $Q_2 \stackrel{\tau^*}{\Longrightarrow} Q_2$ with $(Q'_1, Q_2) \in \mathcal{B}$, or $Q_2 \stackrel{\tau^*}{\Longrightarrow} \bar{Q}_2 \xrightarrow{a} Q'_2$ with $(Q_1, \bar{Q}_2) \in \mathcal{B}$ and $(Q'_1, Q'_2) \in \mathcal{B}$, hence $Q_2 \stackrel{\tau^*}{\Longrightarrow} \xrightarrow{a} \stackrel{\tau^*}{\Longrightarrow} Q'_2$ with $(Q'_1, Q'_2) \in \mathcal{B}$.
  - In the backward case – which cannot be the one of $(P_1, P_2) \in \mathcal{B}$ as both processes are initial – assume that $Q'_1 \xrightarrow{a} Q_1$. There are two subcases:
    * Suppose that Q'_1 is P_1. Then either $a = \tau$ and $(Q'_1, Q_2) \in \mathcal{B}$, where Q_2 is P_2 and $Q_2 \stackrel{\tau^*}{\Longrightarrow} Q_2$, or $Q'_2 \stackrel{\tau^*}{\Longrightarrow} \bar{Q}_2 \xrightarrow{a} Q_2$ with $(Q'_1, \bar{Q}_2) \in \mathcal{B}$ and $(Q'_1, Q'_2) \in \mathcal{B}$, where Q'_2 is P_2 and $Q'_2 \stackrel{\tau^*}{\Longrightarrow} \xrightarrow{a} \stackrel{\tau^*}{\Longrightarrow} Q_2$.
    * If Q'_1 is not P_1, then P_1 reaches Q'_1 with a sequence of moves that are $\mathcal{B}$-compatible with those with which P_2 reaches some Q'_2 such that $(Q'_1, Q'_2) \in \mathcal{B}$ as $\mathcal{B}$ only contains all the processes reachable from P_1 and P_2. Therefore either $a = \tau$ and $(Q_1, Q'_2) \in \mathcal{B}$, where Q'_2 is Q_2 and $Q_2 \stackrel{\tau^*}{\Longrightarrow} Q_2$, or $Q'_2 \stackrel{\tau^*}{\Longrightarrow} \bar{Q}_2 \xrightarrow{a} Q_2$ with $(Q'_1, \bar{Q}_2) \in \mathcal{B}$ in addition to $(Q'_1, Q'_2) \in \mathcal{B}$ and $(Q_1, Q_2) \in \mathcal{B}$, where $Q'_2 \stackrel{\tau^*}{\Longrightarrow} \xrightarrow{a} \stackrel{\tau^*}{\Longrightarrow} Q_2$. ■

According to the logical characterizations of branching bisimilarity shown in [9], this result opens the way to further logical characterizations of $\approx_{\rm FRB}$ over initial processes in addition to the one of Section 3 based on forward and backward modalities:

- The first additional characterization replaces the two aforementioned modalities with an until operator $\phi_1 \langle\langle a \rangle\rangle \phi_2$. This is satisfied by a process P iff either $a = \tau$ with P satisfying ϕ_2, or $P \xRightarrow{\tau^*} \bar{P} \xrightarrow{a} P'$ with every process along $P \xRightarrow{\tau^*} \bar{P}$ satisfying ϕ_1 and P' satisfying ϕ_2.
- The second additional characterization is given by the temporal logic CTL* without the next operator, thanks to a revisitation of the stuttering equivalence of [5] and the bridge between Kripke structures (in which states are labeled with propositions) and labeled transition systems (in which transitions are labeled with actions) built in [9].

5 Conclusion

In this paper we have investigated modal logic characterizations of forward, reverse, and forward-reverse bisimilarities, both strong and weak, over nondeterministic reversible sequential processes. While previous work [4, 3] has addressed compositionality and axiomatizations of those bisimilarities, here the focus has been on identifying suitable modal logics, which are essentially variants of the Hennessy-Milner logic [11], such that two processes are bisimilar iff they satisfy the same set of formulas of the corresponding modal logic.

The additional backward modalities used in this paper are inspired by those in [8], with the important difference that bisimilarities and modal interpretations in the former are defined over states – as is usual – while those in the latter are defined over computation paths. The modal logic characterizations have revealed that strong and weak reverse bisimilarities respectively boil down to strong and weak reverse trace equivalences. Moreover, we have shown that weak forward-reverse bisimilarity coincides with branching bisimilarity [10] over initial processes, thus providing two further logical characterizations for the former thanks to [9].

The study carried out in this paper can contribute, together with the results in [4, 3], to the development of a fully-fledged process algebraic theory of reversible systems. On a more applicative side, following [6] we also observe that the established modal logic characterizations are useful to provide diagnostic information because, whenever two processes are not bisimilar, then there exists at least one formula in the modal logic corresponding to the considered bisimilarity that is satisfied by only one of the two processes and hence can explain the inequivalence.

Acknowledgments. This research has been supported by the PRIN project *NiRvAna – Noninterference and Reversibility Analysis in Private Blockchains*.

References

[1] L. Aceto, A. Ingolfsdottir, K.G. Larsen & J. Srba (2007): *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, doi:10.1017/CBO9780511814105.

[2] C.H. Bennett (1973): *Logical Reversibility of Computation*. *IBM Journal of Research and Development* 17, pp. 525–532, doi:10.1147/rd.176.0525.

[3] M. Bernardo & A. Esposito (2023): *On the Weak Continuation of Reverse Bisimilarity vs. Forward Bisimilarity*. In: *Proc. of the 24th Italian Conf. on Theoretical Computer Science (ICTCS 2023)*, CEUR-WS. To appear.

[4] M. Bernardo & S. Rossi (2023): *Reverse Bisimilarity vs. Forward Bisimilarity*. In: *Proc. of the 26th Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS 2023)*, *LNCS* 13992, Springer, pp. 265–284, doi:10.1007/978-3-031-30829-1_13.

[5] M.C. Browne, E.M. Clarke & O. Grümberg (1988): *Characterizing Finite Kripke Structures in Propositional Temporal Logic*. *Theoretical Computer Science* 59, pp. 115–131, doi:10.1016/0304-3975(88)90098-9.

[6] R. Cleaveland (1990): *On Automatically Explaining Bisimulation Inequivalence*. In: *Proc. of the 2nd Int. Workshop on Computer Aided Verification (CAV 1990)*, *LNCS* 531, Springer, pp. 364–372, doi:10.1007/BFb0023750.

[7] V. Danos & J. Krivine (2004): *Reversible Communicating Systems*. In: *Proc. of the 15th Int. Conf. on Concurrency Theory (CONCUR 2004)*, *LNCS* 3170, Springer, pp. 292–307, doi:10.1007/978-3-540-28644-8_19.

[8] R. De Nicola, U. Montanari & F. Vaandrager (1990): *Back and Forth Bisimulations*. In: *Proc. of the 1st Int. Conf. on Concurrency Theory (CONCUR 1990)*, *LNCS* 458, Springer, pp. 152–165, doi:10.1007/BFb0039058.

[9] R. De Nicola & F. Vaandrager (1995): *Three Logics for Branching Bisimulation*. *Journal of the ACM* 42, pp. 458–487, doi:10.1145/201019.201032.

[10] R.J. van Glabbeek & W.P. Weijland (1996): *Branching Time and Abstraction in Bisimulation Semantics*. *Journal of the ACM* 43, pp. 555–600, doi:10.1145/233551.233556.

[11] M. Hennessy & R. Milner (1985): *Algebraic Laws for Nondeterminism and Concurrency*. *Journal of the ACM* 32, pp. 137–162, doi:10.1145/2455.2460.

[12] M. Hennessy & C. Stirling (1985): *The Power of the Future Perfect in Program Logics*. *Information and Control* 67, pp. 23–52, doi:10.1016/S0019-9958(85)80025-5.

[13] R. Landauer (1961): *Irreversibility and Heat Generation in the Computing Process*. *IBM Journal of Research and Development* 5, pp. 183–191, doi:10.1147/rd.53.0183.

[14] I. Lanese, D. Medić & C.A. Mezzina (2021): *Static versus Dynamic Reversibility in CCS*. *Acta Informatica* 58, pp. 1–34, doi:10.1007/s00236-019-00346-6.

[15] O. Lichtenstein, A. Pnueli & L. Zuck (1985): *The Glory of the Past*. In: *Proc. of the Conf. on Logics in Programs*, *LNCS* 193, Springer, pp. 196–218, doi:10.1007/3-540-15648-8_16.

[16] R. Milner (1989): *Communication and Concurrency*. Prentice Hall.

[17] D. Park (1981): *Concurrency and Automata on Infinite Sequences*. In: *Proc. of the 5th GI Conf. on Theoretical Computer Science*, *LNCS* 104, Springer, pp. 167–183, doi:10.1007/BFb0017309.

[18] I. Phillips & I. Ulidowski (2007): *Reversing Algebraic Process Calculi*. *Journal of Logic and Algebraic Programming* 73, pp. 70–96, doi:10.1016/j.jlap.2006.11.002.

TSO Games
On the decidability of safety games under the total store order semantics

Stephan Spengler
Uppsala University
Uppsala, Sweden
stephan.spengler@it.uu.se

Sanchari Sil
Chennai Mathematical Institute
Chennai, India
sanchari@cmi.ac.in

We consider an extension of the classical Total Store Order (TSO) semantics by expanding it to turn-based 2-player safety games. During her turn, a player can select any of the communicating processes and perform its next transition. We consider different formulations of the safety game problem depending on whether one player or both of them transfer messages from the process buffers to the shared memory. We give the complete decidability picture for all the possible alternatives.

1 Introduction

Most modern architectures, such as Intel x86 [21], SPARC [27], IBM's POWER [20], and ARM [11], implement several relaxations and optimisations that reduce the latency of memory accesses. This has the effect of breaking the Sequential Consistency (SC) assumption [22]. SC is the classical strong semantics for concurrent programs that interleaves the parallel executions of processes while maintaining the order in which instructions were issued. Programmers usually assume that the execution of programs follows the SC model. However, this is not true when we consider concurrent programs running on modern architectures. In fact, even simple programs such as mutual exclusion and producer-consumer protocols, that are correct under SC, may exhibit erroneous behaviors. This is mainly due to the relaxation of the execution order of the instructions. For instance, a standard relaxation is to allow the reordering of reads and writes of the same process if the reads have been issued after the writes and they concern different memory locations. This relaxation can be implemented using an unbounded perfect FIFO queue/buffer between each process and the memory. These buffers are used to store delayed writes. The corresponding model is called Total Store Ordering (TSO) and corresponds to the formalisation of SPARC and Intel x86 [24, 26].

In TSO, an unbounded buffer is associated with each process. When a process executes a write operation, this write is appended to the end of the buffer of that process. A pending write operation on the variable x at the head of a buffer can be deleted in a non-deterministic manner. This updates the value of the shared variable x in the memory. To perform a read operation on a variable x, the process first checks its buffer for a pending write operation on the variable x. If such a write exists, then the process reads the value written by the newest pending write operation on x. Otherwise, the process fetches the value of the variable x from the memory. The verification of programs running under TSO is challenging due to the unboundedness of the buffers. In fact, the induced state space of a program under TSO maybe infinite even if the program itself is a finite-state system.

The reachability problem for programs under TSO checks whether a given program state is reachable during program execution. It is also called safety problem, in case the target state is considered to be a bad state. It has been shown decidable using different alternative semantics for TSO (e.g., [13, 4, 3]). Furthermore, it has been shown in [13] that lossy channel systems (see e.g., [10, 18, 9, 25]) can be simulated by programs running under TSO. This entails that the reachability problem for programs under TSO is non-primitive recursive and that the repeated reachability problem is undecidable. This is an immediate consequence of the fact that the reachability problem for lossy channel is non-primitive recursive [25] and that the repeated reachability problem is undecidable [9]. The termination problem for programs running under TSO has been shown to be decidable in [12] using the framework of well-structured transition systems [18, 10].

The authors of [14, 15] consider the robustness problem for programs running under TSO. This problem consists in checking whether, for any given TSO execution, there is an equivalent SC execution of the same program. Two executions are declared equivalent by the robustness criterion if they agree on (1) the order in which instructions are executed within the same process (i.e., program order), (2) the write instruction from which each read instruction fetches its value (i.e., read-from relation), and (3) the order in which write instruction on the same variable are committed to memory (i.e., store ordering). The problem of checking whether a program is robust has been shown to be PSPACE-complete in [14]. A variant of the robustness problem which is called persistence, declares that two runs are equivalent if (1) they have the same program order and (2) all write instructions reach the memory in the same order. Checking the persistency of a program under TSO has been shown to be PSPACE-complete in [6]. Observe that the persistency and robustness problems are stronger than the safety problem (i.e., if a program is safe under SC and robust/persistent, then it is also safe under TSO).

Due to the non-determinism of the buffer updates, the buffers associated with each process under TSO appear to exhibit a lossy behaviour. Previously, games on lossy channel systems (and more general on monotonic systems) were studied in [8]. Unfortunately these results are not applicable / transferable to programs under TSO whose induced transition systems are not monotone [13].

In this paper, we consider a natural continuation of the works on both the study of the decidability/complexity of the formal verification of programs under TSO and the study of games on concurrent systems. This is further motivated by the fact that formal games provide a framework to reason about a system's behaviour, which can be leveraged in control model checking, for example in controller synthesis problems.

In more detail, we consider (safety) games played on the transition systems induced by programs running under TSO. Given a program under TSO, we construct a game in which two players A and B take turns in executing instructions of the program. The goal of player B is to reach a given set of final configurations, while player A tries to avoid this. Thus, it can also be seen as a reachability game with respect to player B. In this game, the turn determines which player will execute the next program instruction. However, this definition leaves the control of updates undefined. To address this, we give the player the possibility to update memory by removing the pending writes from the buffer between the execution of two instructions.

The control over the buffer updates is shared between the two players in varying ways. We differentiate between multiple scenarios based on when exactly each player is allowed to update. In particular, for each player A or B we have the following cases: (1) she can never update, (2) she can update after her own turn, (3) she can update before her own turn, and (4) she can always update, i.e. before and after her own turn. In total, we obtain an exhaustive collection of 16 different TSO games. We divide these 16 games into four different groups, depending on their decidability results.

- Group I (7 games) can be reduced to TSO games with 2-bounded buffers.
- Group II (1 game) can be reduced to TSO games with bounded buffers.
- Group III (7 games) can simulate perfect channel systems.
- Group IV (1 game) can be reduced to a finite game without buffers.

		Player A:			
		always	before	after	never
	always	I (d)			
Player B:	before		II (d)		
	after			III (u)	
	never				IV (d)

Figure 1: Groups of TSO games, where players A and B are allowed to update the buffer: always, before their own move, after their own move, or never. The games in group I, II and IV are decidable (d), the games in group III are undecidable (u).

This classification is shown in Figure 1. Of these four groups, only Group III is undecidable, the others each reduce to a finite game and are thus decidable.

Finally, we establish the exact computational complexity for the decidable games. In fact, we show that the problem is EXPTIME-complete. We prove EXPTIME-hardness by a reduction from the problem of acceptance of a word by a linearly bounded alternating Turing machine [17]. To prove EXPTIME-membership, we show that it is possible to compute the set of winning region for player B in exponential time. These results are surprising given the non-primitive recursive complexity of the reachability problem for programs under TSO and the undecidability of the repeated reachability problem.

Related Works. In addition to the related work mentioned in the introduction on the decidability / complexity of the verification problems of programs running under TSO, there have been some works on parameterized verification of programs running under TSO. The problem consists in verifying a concurrent program regardless of the number of involved processes (which are identical finite-state systems). The parameterised reachability problem of programs running under TSO has been shown to be decidable in [2, 3]. While this problem for concurrent programs performing only read and writing operations (no atomic read-write instructions) is PSPACE-complete [7]. This result has been recently extended to processes manipulating abstract data types over infinite domains [5]. Checking the robustness of a parameterised concurrent system is decidable and EXPSPACE-hard [14].

As far as we know this is the first work that considers the game problem for programs running under TSO. The proofs and techniques used in this paper are different from the ones used to prove decidability / complexity results for the verification of programs under TSO except the undecidability result which uses some ideas from the reduction from the reachability problem for lossy channel systems to its corresponding problem for programs under TSO [13]. However, our undecidability proof requires us to implement a protocol that detects lossiness of messages in order to turn the lossy channel system into a perfect one (which is the most intricate part of the proof).

2 Preliminaries

2.1 Transition Systems

A *(labeled) transition system* is a triple $\langle \mathsf{C}, \mathsf{L}, \rightarrow \rangle$, where C is a set of *configurations*, L is a set of *labels*, and $\rightarrow \subseteq \mathsf{C} \times \mathsf{L} \times \mathsf{C}$ is a *transition relation*. We usually write $\mathsf{c}_1 \xrightarrow{\mathsf{label}} \mathsf{c}_2$ if $\langle \mathsf{c}_1, \mathsf{label}, \mathsf{c}_2 \rangle \in \rightarrow$. Furthermore, we write $\mathsf{c}_1 \rightarrow \mathsf{c}_2$ if there exists some label such that $\mathsf{c}_1 \xrightarrow{\mathsf{label}} \mathsf{c}_2$. A *run* π of $\mathcal{T}$ is a sequence of transitions $\mathsf{c}_0 \xrightarrow{\mathsf{label}_1} \mathsf{c}_1 \xrightarrow{\mathsf{label}_2} \mathsf{c}_2 \ldots \xrightarrow{\mathsf{label}_n} \mathsf{c}_n$. It is also written as $\mathsf{c}_0 \xrightarrow{\pi} \mathsf{c}_n$. A configuration c' is *reachable* from a configuration c, if there exists a run from c to c'.

For a configuration c, we defined $\text{Pre}(\mathsf{c}) = \{\mathsf{c}' \mid \mathsf{c}' \rightarrow \mathsf{c}\}$ and $\text{Post}(\mathsf{c}) = \{\mathsf{c}' \mid \mathsf{c} \rightarrow \mathsf{c}'\}$. We extend these notions to sets of configurations C' with $\text{Pre}(\mathsf{C}') = \bigcup_{\mathsf{c} \in \mathsf{C}'} \text{Pre}(\mathsf{c})$ and $\text{Post}(\mathsf{C}') = \bigcup_{\mathsf{c} \in \mathsf{C}'} \text{Post}(\mathsf{c})$.

An *unlabeled transition system* is a transition system without labels. Formally, it is defined as a TS with a singleton label set. In this case, we omit the labels.

2.2 Perfect Channel Systems

Given a set of messages M, define the set of channel operations $\mathsf{Op} = \{!\mathsf{m}, ?\mathsf{m} \mid \mathsf{m} \in \mathsf{M}\} \cup \{\texttt{skip}\}$. A *perfect channel system* (PCS) is a triple $\mathcal{L} = \langle \mathsf{S}, \mathsf{M}, \delta \rangle$, where S is a set of states, M is a set of messages, and $\delta \subseteq \mathsf{S} \times \mathsf{Op} \times \mathsf{S}$ is a transition relation. We write $\mathsf{s}_1 \xrightarrow{\mathsf{op}} \mathsf{s}_2$ if $\langle \mathsf{s}_1, \mathsf{op}, \mathsf{s}_2 \rangle \in \delta$.

Intuitively, a PCS models a finite state automaton that is augmented by a *perfect* (i.e. non-lossy) FIFO buffer, called *channel*. During a *send operation* $!\mathsf{m}$, the channel system appends m to the tail of the channel. A transition $?\mathsf{m}$ is called *receive operation*. It is only enabled if the channel is not empty and m is its oldest message. When the channel system performs this operation, it removes m from the head of the channel. Lastly, a `skip` operation just changes the state, but does not modify the buffer.

The formal semantics of $\mathcal{L}$ are defined by a transition system $\mathcal{T}_{\mathcal{L}} = \langle \mathsf{C}_{\mathcal{L}}, \mathsf{L}_{\mathcal{L}}, \rightarrow_{\mathcal{L}} \rangle$, where $\mathsf{C}_{\mathcal{L}} = \mathsf{S} \times \mathsf{M}^*$, $\mathsf{L}_{\mathcal{L}} = \mathsf{Op}$ and the transition relation $\rightarrow_{\mathcal{L}}$ is the smallest relation given by:

- If $\mathsf{s}_1 \xrightarrow{!\mathsf{m}} \mathsf{s}_2$ and $\mathsf{w} \in \mathsf{M}^*$, then $\langle \mathsf{s}_1, \mathsf{w} \rangle \xrightarrow{!\mathsf{m}}_{\mathcal{L}} \langle \mathsf{s}_2, \mathsf{m} \cdot \mathsf{w} \rangle$.
- If $\mathsf{s}_1 \xrightarrow{?\mathsf{m}} \mathsf{s}_2$ and $\mathsf{w} \in \mathsf{M}^*$, then $\langle \mathsf{s}_1, \mathsf{w} \cdot \mathsf{m} \rangle \xrightarrow{?\mathsf{m}}_{\mathcal{L}} \langle \mathsf{s}_2, \mathsf{w} \rangle$.
- If $\mathsf{s}_1 \xrightarrow{\texttt{skip}} \mathsf{s}_2$ and $\mathsf{w} \in \mathsf{M}^*$, then $\langle \mathsf{s}_1, \mathsf{w} \rangle \xrightarrow{\texttt{skip}}_{\mathcal{L}} \langle \mathsf{s}_2, \mathsf{w} \rangle$.

A state $\mathsf{s}_F \in \mathsf{S}$ is *reachable* from a configuration $\mathsf{c}_0 \in \mathsf{C}_{\mathcal{L}}$, if there exists a configuration $\mathsf{c}_F = \langle \mathsf{s}_F, \mathsf{w}_F \rangle$ such that c_F is reachable from c_0 in $\mathcal{T}_{\mathcal{L}}$. The **state reachability problem** of PCS is, given a perfect channel system $\mathcal{L}$, an initial configuration $\mathsf{c}_0 \in \mathsf{C}_{\mathcal{L}}$ and a final state $\mathsf{s}_F \in \mathsf{S}$, to decide whether s_F is reachable from c_0 in $\mathcal{T}_{\mathcal{L}}$. It is undecidable [16].

3 Concurrent Programs

3.1 Syntax

Let Dom be a finite data domain and Vars be a finite set of shared variables over Dom. We define the *instruction set* $\mathsf{Instrs} = \{\texttt{rd}(\mathsf{x}, \mathsf{d}), \texttt{wr}(\mathsf{x}, \mathsf{d}) \mid \mathsf{x} \in \mathsf{Vars}, \mathsf{d} \in \mathsf{Dom}\} \cup \{\texttt{skip}, \texttt{mf}\}$, which are called *read*, *write*, *skip* and *memory fence*, respectively. A process is represented by a finite state labeled transition system. It is given as the triple $\mathsf{Proc} = \langle \mathsf{Q}, \mathsf{Instrs}, \delta \rangle$, where Q is a finite set of *local states* and $\delta \subseteq \mathsf{Q} \times \mathsf{Instrs} \times \mathsf{Q}$ is the transition relation. As with transition systems, we write $\mathsf{q}_1 \xrightarrow{\mathsf{instr}} \mathsf{q}_2$ if $\langle \mathsf{q}_1, \mathsf{instr}, \mathsf{q}_2 \rangle \in \delta$ and $\mathsf{q}_1 \rightarrow \mathsf{q}_2$ if there exists some instr such that $\mathsf{q}_1 \xrightarrow{\mathsf{instr}} \mathsf{q}_2$.

A *concurrent program* is a tuple of processes $\mathcal{P} = \langle \mathsf{Proc}^\iota \rangle_{\iota \in \mathcal{I}}$, where $\mathcal{I}$ is a finite set of process identifiers. For each $\iota \in \mathcal{I}$ we have $\mathsf{Proc}^\iota = \langle \mathsf{Q}^\iota, \mathsf{Instrs}, \delta^\iota \rangle$. A *global* state of $\mathcal{P}$ is a function $\mathcal{S} : \mathcal{I} \rightarrow \bigcup_{\iota \in \mathcal{I}} \mathsf{Q}^\iota$ that maps each process to its local state, i.e $\mathcal{S}(\iota) \in \mathsf{Q}^\iota$.

3.2 TSO Semantics

Under TSO semantics, the processes of a concurrent program do not interact with the shared memory directly, but indirectly through a FIFO *store buffer* instead. When performing a *write* instruction $\texttt{wr}(\mathsf{x}, \mathsf{d})$,

read-own-write
$$\frac{\mathsf{q} \xrightarrow{\texttt{rd}(\mathsf{x},\mathsf{d})} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q} \qquad \mathcal{B}(\iota)|_{\{\mathsf{x}\}\times\mathsf{Dom}}=\langle \mathsf{x},\mathsf{d}\rangle\cdot\mathsf{w}}{\langle \mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\texttt{rd}(\mathsf{x},\mathsf{d})_\iota}_{\mathcal{P}} \langle \mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B},\mathcal{M}\rangle}$$

read-from-memory
$$\frac{\mathsf{q} \xrightarrow{\texttt{rd}(\mathsf{x},\mathsf{d})} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q} \qquad \mathcal{B}(\iota)|_{\{\mathsf{x}\}\times\mathsf{Dom}}=\varepsilon \qquad \mathcal{M}(\mathsf{x})=\mathsf{d}}{\langle \mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\texttt{rd}(\mathsf{x},\mathsf{d})_\iota}_{\mathcal{P}} \langle \mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B},\mathcal{M}\rangle}$$

write
$$\frac{\mathsf{q} \xrightarrow{\texttt{wr}(\mathsf{x},\mathsf{d})} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q}}{\langle \mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\texttt{wr}(\mathsf{x},\mathsf{d})_\iota}_{\mathcal{P}} \langle \mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B}[\iota\leftarrow\langle \mathsf{x},\mathsf{d}\rangle\cdot\mathcal{B}(\iota)],\mathcal{M}\rangle}$$

skip
$$\frac{\mathsf{q} \xrightarrow{\texttt{skip}} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q}}{\langle \mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\texttt{skip}_\iota}_{\mathcal{P}} \langle \mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B},\mathcal{M}\rangle}$$

memory-fence
$$\frac{\mathsf{q} \xrightarrow{\texttt{mf}} \mathsf{q}' \qquad \mathcal{S}(\iota)=\mathsf{q} \qquad \mathcal{B}(\iota)=\varepsilon}{\langle \mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\texttt{mf}_\iota}_{\mathcal{P}} \langle \mathcal{S}[\iota\leftarrow\mathsf{q}'],\mathcal{B},\mathcal{M}\rangle}$$

update
$$\frac{\mathcal{B}(\iota)=\mathsf{w}\cdot\langle \mathsf{x},\mathsf{d}\rangle}{\langle \mathcal{S},\mathcal{B},\mathcal{M}\rangle \xrightarrow{\texttt{up}_\iota}_{\mathcal{P}} \langle \mathcal{S},\mathcal{B}[\iota\leftarrow\mathsf{w}],\mathcal{M}[\mathsf{x}\leftarrow\mathsf{d}]\rangle}$$

Figure 2: TSO semantics

the process adds a new message $\langle \mathsf{x},\mathsf{d}\rangle$ to the tail of its store buffer. A *read* instruction $\texttt{rd}(\mathsf{x},\mathsf{d})$ works differently depending on the current buffer content of the process. If the buffer contains a write message on variable x, the value d must correspond to the value of the most recent such message. Otherwise, the value is read directly from memory. A *skip* instruction only changes the local state of the process. The *memory fence* instruction is disabled, i.e. it cannot be executed, unless the buffer of the process is empty. Additionally, at any point during the execution, the process can *update* the write message at the head of its buffer to the memory. For example, if the oldest message in the buffer is $\langle \mathsf{x},\mathsf{d}\rangle$, it will be removed from the buffer and the memory value of variable x will be updated to contain the value d. This happens in a non-deterministic manner.

Formally, we introduce a TSO *configuration* as a tuple $\mathsf{c} = \langle \mathcal{S},\mathcal{B},\mathcal{M}\rangle$, where:

- $\mathcal{S} : \mathcal{I} \to \bigcup_{\iota\in\mathcal{I}} \mathsf{Q}^\iota$ is a global state of $\mathcal{P}$.
- $\mathcal{B} : \mathcal{I} \to (\mathsf{Vars}\times\mathsf{Dom})^*$ represents the buffer state of each process.
- $\mathcal{M} : \mathsf{Vars} \to \mathsf{Dom}$ represents the memory state of each shared variable.

Given a configuration c, we write $\mathcal{S}(\mathsf{c})$, $\mathcal{B}(\mathsf{c})$ and $\mathcal{M}(\mathsf{c})$ for the global program state, buffer state and memory state of c. The semantics of a concurrent program running under TSO is defined by a transition system $\mathcal{T}_{\mathcal{P}} = \langle \mathsf{C}_{\mathcal{P}},\mathsf{L}_{\mathcal{P}},\to_{\mathcal{P}}\rangle$, where $\mathsf{C}_{\mathcal{P}}$ is the set of all possible TSO configurations, $\mathsf{L}_{\mathcal{P}} = \{\mathsf{instr}_\iota \mid \mathsf{instr}\in\mathsf{Instrs}, \iota\in\mathcal{I}\} \cup \{\mathsf{up}_\iota \mid \iota\in\mathcal{I}\}$ is the set of labels. The transition relation $\to_{\mathcal{P}}$ is given by the rules in Figure 2, where we use $\mathcal{B}(\iota)|_{\{\mathsf{x}\}\times\mathsf{Dom}}$ to denote the restriction of $\mathcal{B}(\iota)$ to write messages on the variable x.

A global state $\mathcal{S}_F$ is *reachable* from an initial configuration c_0, if there is a configuration c_F with $\mathcal{S}(\mathsf{c}_F) = \mathcal{S}_F$ such that c_F is reachable from c_0 in $\mathcal{T}_{\mathcal{P}}$. The **state reachability problem** of TSO is, given a program $\mathcal{P}$, an initial configuration c_0 and a final global state $\mathcal{S}_F$, to decide whether $\mathcal{S}_F$ is reachable from c_0 in $\mathcal{T}_{\mathcal{P}}$.

We define up^* to be the transitive closure of $\{\mathrm{up}_\iota \mid \iota \in \mathcal{I}\}$, i.e. $\mathsf{c}_1 \xrightarrow{\mathrm{up}^*}_{\mathcal{P}} \mathsf{c}_2$ if and only if c_2 can be obtained from c_1 by some amount of buffer updates.

4 Games

4.1 Definitions

A *(safety) game* is an unlabeled transition sytem, in which two players A and B take turns making a *move* in the transition system, i.e. changing the state of the game from one configuration to an adjacent one. The goal of player B is to reach a given set of final configurations, while player A tries to avoid this. Thus, it can also be seen as a *reachability* game with respect to player B.

Formally, a game is defined as a tuple $\mathcal{G} = \langle \mathsf{C}, \mathsf{C}_A, \mathsf{C}_B, \rightarrow, \mathsf{C}_F \rangle$, where C is the set of configurations, C_A and C_B form a partition of C, the transition relation is restricted to $\rightarrow \subseteq (\mathsf{C}_A \times \mathsf{C}_B) \cup (\mathsf{C}_B \times \mathsf{C}_A)$, and $\mathsf{C}_F \subseteq \mathsf{C}_A$ is a set of *final states*. Furthermore, we assume without loss of generality that $\mathcal{G}$ is deadlock-free, i.e. $\mathrm{Post}(\mathsf{c}) \neq \emptyset$ for all $\mathsf{c} \in \mathsf{C}$.

A *play* P of $\mathcal{G}$ is an infinite sequence $\mathsf{c}_0, \mathsf{c}_1, \ldots$ such that $\mathsf{c}_i \rightarrow \mathsf{c}_{i+1}$ for all $i \in \mathbb{N}$. In the context of safety games, P is *winning* for player B if there is $i \in \mathbb{N}$ such that $\mathsf{c}_i \in \mathsf{C}_F$. Otherwise, it is *winning* for player A. This means that player B tries to force the play into C_F, while player A tries to avoid this.

A *strategy* of player A is a partial function $\sigma_A : \mathsf{C}^* \rightharpoonup \mathsf{C}_B$, such that $\sigma_A(\mathsf{c}_0, \ldots, \mathsf{c}_n)$ is defined if and only if $\mathsf{c}_0, \ldots, \mathsf{c}_n$ is a prefix of a play, $\mathsf{c}_n \in \mathsf{C}_A$ and $\sigma_A(\mathsf{c}_0, \ldots, \mathsf{c}_n) \in \mathrm{Post}(\mathsf{c}_n)$. A strategy σ_A is called *positional*, if it only depends on c_n, i.e. if $\sigma_A(\mathsf{c}_0, \ldots, \mathsf{c}_n) = \sigma_A(\mathsf{c}_n)$ for all $(\mathsf{c}_0, \ldots, \mathsf{c}_n)$ on which σ_A is defined. Thus, a positional strategy is usually given as a total function $\sigma_A : \mathsf{C}_A \rightarrow \mathsf{C}_B$. Given two games $\mathcal{G}$ and $\mathcal{G}'$ and a strategy σ_A for $\mathcal{G}$, an *extension* of σ_A to $\mathcal{G}'$ is a strategy σ'_A of $\mathcal{G}'$ that is also an extension of σ_A to the configuration set of $\mathcal{G}'$ in the mathematical sense, i.e. $\sigma_A(\mathsf{c}_0, \ldots, \mathsf{c}_n) = \sigma'_A(\mathsf{c}_0, \ldots, \mathsf{c}_n)$ for all $(\mathsf{c}_0, \ldots, \mathsf{c}_n)$ on which σ_A is defined. Conversely, σ_A is called the *restriction* of σ'_A to $\mathcal{G}$. For player B, strategies are defined accordingly.

Two strategies σ_A and σ_B together with an initial configuration c_0 induce a play $\mathsf{P}(\mathsf{c}_0, \sigma_A, \sigma_B) = \mathsf{c}_0, \mathsf{c}_1, \ldots$ such that $\mathsf{c}_{i+1} = \sigma_A(\mathsf{c}_0, \ldots, \mathsf{c}_i)$ for all $\mathsf{c}_i \in \mathsf{C}_A$ and $\mathsf{c}_{i+1} = \sigma_B(\mathsf{c}_0, \ldots, \mathsf{c}_i)$ for all $\mathsf{c}_i \in \mathsf{C}_B$. A strategy σ_A is *winning* from a configuration c, if for *all* strategies σ_B it holds that $\mathsf{P}(\sigma_A, \sigma_B, \mathsf{c})$ is a winning play for player A. A configuration c is *winning* for player A if she has a strategy that is winning from c. Equivalent notions exist for player B. The **safety problem** for a game $\mathcal{G}$ and a configuration c is to decide whether c is winning for player A.

Lemma 1 (Proposition 2.21 in [23])**.** *In safety games, every configuration is winning for exactly one player. A player with a winning strategy also has a positional winning strategy.*

Since we only consider safety games in this paper, strategies will be considered to be positional unless explicitly stated otherwise. Furthermore, Lemma 1 implies the following:

- $\mathsf{c}_A \in \mathsf{C}_A$ is winning for player A $\iff$ there is $\mathsf{c}_B \in \mathrm{Post}(\mathsf{c}_A)$ that is winning for player A.
- $\mathsf{c}_B \in \mathsf{C}_B$ is winning for player A $\iff$ all $\mathsf{c}_A \in \mathrm{Post}(\mathsf{c}_B)$ are winning for player A.

A *finite game* is a game with a finite set of configurations. It is rather intuitive that the safety problem is decidable for finite games, e.g. by applying a backward induction algorithm. In particular, the winning configurations for each player are computable in linear time:

Lemma 2 (Chapter 2 in [19])**.** *Computing the set of winning configurations for a finite game with n configurations and m transitions is in $\mathcal{O}(n+m)$.*

4.2 TSO games

A TSO program $\mathcal{P} = \langle \mathsf{Proc}^{\iota} \rangle_{\iota \in \mathcal{I}}$ and a set of final local states $\mathsf{Q}_F^{\mathcal{P}} \subseteq \mathsf{Q}^{\mathcal{P}}$ induce a safety game $\mathcal{G}(\mathcal{P}, \mathsf{Q}_F^{\mathcal{P}}) = \langle \mathsf{C}, \mathsf{C}_A, \mathsf{C}_B, \rightarrow, \mathsf{C}_F \rangle$ as follows. The sets C_A and C_B are copies of the set $\mathsf{C}^{\mathcal{P}}$ of TSO configurations, annotated by A and B, respectively: $\mathsf{C}_A := \{\mathsf{c}_A \mid \mathsf{c} \in \mathsf{C}^{\mathcal{P}}\}$ and $\mathsf{C}_B := \{\mathsf{c}_B \mid \mathsf{c} \in \mathsf{C}^{\mathcal{P}}\}$. The set of final configurations is defined as $\mathsf{C}_F := \{\langle \mathcal{S}, \mathcal{B}, \mathcal{M} \rangle_A \in \mathsf{C}_A \mid \exists\, \iota \in \mathcal{I} : \mathcal{S}(\iota) \in \mathsf{Q}_F^{\mathcal{P}}\}$, i.e. the set of all configurations where at least one process is in a final state. The transition relation $\rightarrow$ is defined by the following rules:

- For each transition $\mathsf{c} \xrightarrow{\mathsf{instr}_\iota}_{\mathcal{P}} \mathsf{c}'$ where $\mathsf{c}, \mathsf{c}' \in \mathsf{C}^{\mathcal{P}}$, $\iota \in \mathcal{I}$ and $\mathsf{instr} \in \mathsf{Instrs}$, it holds that $\mathsf{c}_A \rightarrow \mathsf{c}'_B$ and $\mathsf{c}_B \rightarrow \mathsf{c}'_A$. This means that each player can execute any TSO instruction, but they take turns alternatingly.
- *If player A can update before her own turn:* For each transition $\mathsf{c}_A \rightarrow \mathsf{c}'_B$ introduced by any of the previous rules, it holds that $\tilde{\mathsf{c}}_A \rightarrow \mathsf{c}'_B$ for all $\tilde{\mathsf{c}}$ with $\tilde{\mathsf{c}} \xrightarrow{\mathrm{up}^*}_{\mathcal{P}} \mathsf{c}$.
- *If player A can update after her own turn:* For each transition $\mathsf{c}_A \rightarrow \mathsf{c}'_B$ introduced by any of the previous rules, it holds that $\mathsf{c}_A \rightarrow \tilde{\mathsf{c}}'_B$ for all $\tilde{\mathsf{c}}'$ with $\mathsf{c}' \xrightarrow{\mathrm{up}^*}_{\mathcal{P}} \tilde{\mathsf{c}}'$.
- The update rules for player B are defined in a similar manner.

From this definition, we obtain 16 different variants of TSO games, which differ in whether each of the players can update *never*, *before* her turn, *after* her turn, or *always* (before and after her turn). We group games with similar decidability and complexity results together. An overview of these four groups is presented in Figure 1. Each group is described in detail in the following sections.

But first, we present a general result that gives a lower complexity bound for all groups of TSO games. Unexpectedly, even a single process is enough to show EXPTIME-hardness. We prove this by reducing the *word acceptance problem* of *linearly bounded alternating Turing machines* (ATM) to the safety problem of a single-process TSO game. The idea is to store the state and head position of the ATM in the local state of the process, and use a set of variables to save the word on the working tape. Based on the alternations of the Turing machine, either player A or player B decides which transition the program will simulate. Interestingly, we can argue that the exact type of TSO game is irrelevant. Moreover, the construction does not make use of the memory buffers, which implies that the result would even hold if the program followed SC semantics. The formal proof can be found in Appendix A of the extended version of this paper [28].

Theorem 3. *The safety problem for TSO games is* EXPTIME*-hard.*

5 Group I

All TSO games in this group have the following in common: There is one player that can update messages *after* her turn, and the other player can update messages *before* her turn. Both players might be allowed to do more than that, but fortunately we do not need to differentiate between those cases. In the following, we call the player that updates after her turn *player X*, and the other one *player Y*. Although the definition of safety games seems to be of asymmetric nature (player B tries to *reach* a final configuration, while player A tries to *avoid* them), the proof does not rely on the exact identity of player X and Y.

In this section, given a configuration c, we write $\bar{\mathsf{c}}$ to denote the unique configuration obtained from c after updating all messages to the memory. More formally, $\mathsf{c} \xrightarrow{\mathrm{up}^*} \bar{\mathsf{c}}$ and all buffers of $\bar{\mathsf{c}}$ are empty.

Let $\mathcal{G} = \langle \mathsf{C}, \mathsf{C}_A, \mathsf{C}_B, \rightarrow, \mathsf{C}_F \rangle$ be a TSO game as described above, currently in some configuration $\mathsf{c}_0 \in \mathsf{C}$. We first consider the situation where player X has a winning strategy σ_X from c_0. Let σ_Y be an

$$\begin{tikzcd}
\mathsf{c} \arrow[r, "\sigma_X"] \arrow[dr, "\bar{\sigma}_X"'] & \mathsf{c}' \arrow[r, "\bar{\sigma}_Y"] \arrow[d, "\mathrm{up}^*"] & \mathsf{c}'' \\
& \bar{\mathsf{c}}' \arrow[ur, "\sigma_Y"'] &
\end{tikzcd}$$

Figure 3: Commutative diagram of strategies in games of group I.

arbitrary strategy for player Y and define two more strategies $\bar{\sigma}_X : \mathsf{c} \mapsto \overline{\sigma_X(\mathsf{c})}$ and $\bar{\sigma}_Y : \mathsf{c} \mapsto \sigma_Y(\bar{\mathsf{c}})$. That is, they act like σ_X and σ_Y, respectively, with the addition that $\bar{\sigma}_X$ empties the buffer *after* each turn and $\bar{\sigma}_Y$ empties the buffer *before* each turn. From the definitions it follows directly that $\bar{\sigma}_Y(\sigma_X(\mathsf{c})) = \sigma_Y(\bar{\sigma}_X(\mathsf{c}))$ for all $\mathsf{c} \in \mathsf{C}_X$. An example can be seen in Figure 3.

We argue that $\bar{\sigma}_X$ is a winning strategy for player X. The intuition behind this is as follows: Using the notation of Figure 3, if a configuration c'' is reachable from $\bar{\mathsf{c}}'$, then it is also reachable from c', since player Y can empty all buffers at the start of her turn and then proceed as if she started in $\bar{\mathsf{c}}'$. On the other hand, there might be configurations reachable from c' but not $\bar{\mathsf{c}}'$, for example a read transition might get disabled by one of the buffer updates. Thus, player X never gets a disadvantage by emptying the buffers.

Claim 4. *$\bar{\sigma}_X$ is a winning strategy from c_0.*

Proof. **Case** $\mathsf{c}_0 \in \mathsf{C}_X$: Since $\bar{\sigma}_Y(\sigma_X(\mathsf{c})) = \sigma_Y(\bar{\sigma}_X(\mathsf{c}))$ for all $\mathsf{c} \in \mathsf{C}_X$, the plays $\mathsf{P}_1 = \mathsf{P}(\mathsf{c}_0, \sigma_X, \bar{\sigma}_Y)$ and $\mathsf{P}_2 = \mathsf{P}(\mathsf{c}_0, \bar{\sigma}_X, \sigma_Y)$ agree on every second configuration, i.e. the configurations in C_X. Moreover, the configurations in between (after an odd number of steps) at least share the same global state, i.e. $\mathcal{S}(\sigma_X(\mathsf{c})) = \mathcal{S}(\bar{\sigma}_X(\mathsf{c}))$. In particular, the sequence of visited global TSO states is the same in both plays. Since σ_X is a winning strategy from c_0, it means that P_1 is winning for player X. This means that P_2 is also winning, because for both players, a winning play is clearly determined by the sequence of visited global TSO states. Because we chose σ_Y arbitrarily, it follows that $\bar{\sigma}_X$ is a winning strategy.

Case $\mathsf{c}_0 \in \mathsf{C}_Y$: For the other case, we consider the configurations in $\mathrm{Post}(\mathsf{c}_0)$ instead. We observe that σ_X must be a winning strategy for all $\mathsf{c} \in \mathrm{Post}(\mathsf{c}_0)$. We apply the first case of this proof to each of these configurations and obtain that $\bar{\sigma}_X$ is a winning strategy for all of them. It follows that $\bar{\sigma}_X$ is a winning strategy for c_0. □

Suppose that player X plays her modified strategy as described above. We observe that after at most two steps, every play induced by her strategy and an arbitrary strategy of the opposing player only visits configurations with at most one message in the buffers: Player X will empty all buffers at the end of each of her turns and player Y can only add at most one message to the buffers in between. Hence, they can play on a finite set of configurations instead.

To show this, we construct a finite game $\mathcal{G}' = \langle \mathsf{C}', \mathsf{C}'_A, \mathsf{C}'_B, \rightarrow', \mathsf{C}'_F \rangle$ as follows. C'_Y contains all configurations of C_Y that have at most one buffer message, i.e. $\{\langle \mathcal{S}, \mathcal{B}, \mathcal{M} \rangle_Y \in \mathsf{C}_Y \mid \sum_{\iota \in \mathcal{I}} |\mathcal{B}(\iota)| \leq 1\}$. If $\mathsf{c}_0 \in \mathsf{C}_Y$, we also add it to C'_Y, otherwise to C'_X. Lastly, we add $\mathrm{Post}(\mathsf{C}'_Y)$ to C'_X, where Post is with respect to $\mathcal{G}$. $\rightarrow'$ is defined as the restriction of $\rightarrow$ to configurations of $\mathcal{G}'$, and $\mathsf{C}'_F = \mathsf{C}_F \cap \mathsf{C}'_A$. Note that C'_X also contains configurations with two messages. This is needed to account for the case that player Y has a winning strategy, which is handled later in this proof. Now, let $\bar{\sigma}'_X$ be the restriction of $\bar{\sigma}_X$ to C'_X (in the mathematical sense, i.e $\bar{\sigma}'_X : \mathsf{C}'_X \rightarrow \mathsf{C}_Y$ and $\bar{\sigma}_X(\mathsf{c}) = \bar{\sigma}'_X(\mathsf{c})$ for all $\mathsf{c} \in \mathsf{C}'_X$).

Claim 5. *$\bar{\sigma}'_X$ is a winning strategy for c_0 in $\mathcal{G}'$.*

Proof. Looking at the definitions, we confirm that $\bar{\sigma}'_X$ actually is a valid strategy for $\mathcal{G}'$, i.e. $\bar{\sigma}'_X(\mathsf{c}) \in \mathsf{C}'_Y$, for all $\mathsf{c} \in \mathsf{C}'_X$, since $\bar{\sigma}'_X(\mathsf{c})$ has empty buffers. (This makes $\bar{\sigma}'_X$ the restriction of $\bar{\sigma}_X$ to $\mathcal{G}'$.) Consider a strategy σ'_Y for player Y in $\mathcal{G}'$ and an arbitrary extension σ_Y to $\mathcal{G}$. Because $\bar{\sigma}'_X$ and $\bar{\sigma}_X$ agree on C'_X and $\bar{\sigma}'_Y$ and $\bar{\sigma}_Y$ agree on C'_Y, $\mathsf{P} = \mathsf{P}(\mathsf{c}_0, \bar{\sigma}'_X, \bar{\sigma}_Y)$ and $\mathsf{P}' = \mathsf{P}(\mathsf{c}_0, \bar{\sigma}'_X, \bar{\sigma}_Y)$ are in fact the exact same play. Since $\bar{\sigma}_X$ is a winning strategy, P is a winning play, and thus also P′. Here, note that $\mathcal{G}$ and $\mathcal{G}'$ agree on the final configurations within C′. Since σ'_Y was arbitrary, it follows that $\bar{\sigma}'_X$ is a winning strategy from c_0 in $\mathcal{G}'$. □

What is left to show is that a winning strategy for $\mathcal{G}'$ induces a winning strategy for $\mathcal{G}$. Suppose σ'_X is a winning strategy for player X in game $\mathcal{G}'$ for the configuration c_0. Let σ_X be an arbitrary extension of σ'_X to $\mathcal{G}$.

Claim 6. *σ_X is a winning strategy for c_0 in $\mathcal{G}$.*

Proof. Let σ_Y be a strategy of player Y in $\mathcal{G}$ and σ'_Y the restriction of σ_Y to C'_Y (again, in the mathematical sense). Since the outgoing transitions of every $\mathsf{c} \in \mathsf{C}'_Y$ are the same in both $\mathcal{G}$ and $\mathcal{G}'$, σ'_Y is a strategy for $\mathcal{G}'$ (and the restriction of σ_Y to $\mathcal{G}'$). Furthermore, starting from c_0, we see that σ_X and σ_Y induce the exact same play in $\mathcal{G}$ as σ'_X and σ'_Y in $\mathcal{G}'$. Since the former play is winning, so must be the latter one. □

Now, we quickly cover the situation where it is player Y that has a winning strategy. We follow the same arguments as previously, with minor changes. This time, assume σ_Y to be a winning strategy and let σ_X be arbitrary. Define $\bar{\sigma}_X$ and $\bar{\sigma}_Y$ as above. Following the beginning of the proof of Claim 4, we can conclude that the sequence of visited global TSO states is the same in both play P_1 and P_2. For the remainder of the proof, we swap the roles of X and Y and obtain that $\bar{\sigma}_Y$ is a winning strategy.

Let $\bar{\sigma}'_Y$ be the restriction of $\bar{\sigma}_Y$ to C'_Y. Since $\bar{\sigma}'_Y(\mathsf{C}'_Y) = \bar{\sigma}_Y(\mathsf{C}'_Y) \subseteq \mathrm{Post}(\mathsf{C}'_Y) \subseteq \mathsf{C}'_X$, it follows that $\bar{\sigma}'_Y$ is a strategy of $\mathcal{G}'$ (Post is again with respect to $\mathcal{G}$). Consider a strategy σ'_X for player X in $\mathcal{G}'$ and an arbitrary extension σ_X to $\mathcal{G}$. Similar as in Claim 5, we see that $\mathsf{P}(\mathsf{c}_0, \bar{\sigma}'_X, \bar{\sigma}_Y) = \mathsf{P}(\mathsf{c}_0, \bar{\sigma}'_X, \bar{\sigma}_Y)$ and conclude that $\bar{\sigma}'_Y$ is a winning strategy.

The other direction follows from the proof of Claim 6, with the roles of X and Y swapped.

Theorem 7. *The safety problem for games of group I is* EXPTIME*-complete.*

Proof. By Claim 4 and Claim 5, if a configuration c_0 is winning for player X in $\mathcal{G}$, then it is also winning in $\mathcal{G}'$. The reverse holds by Claim 6. The equivalent statement for player Y follows from results outlined above. Thus, the safety problem for $\mathcal{G}$ is equivalent to the safety problem for $\mathcal{G}'$. $\mathcal{G}'$ is finite and has exponentially many configurations. EXPTIME-completeness follows immediately from Lemma 2 (membership) and Theorem 3 (hardness). □

Remark 8. In the game where both players are allowed to update the buffer at any time, we can show an interesting conclusion. By Claim 4 and the equivalent statement for the second player, we can restrict both players to strategies that empty the buffer after each turn. Thus, the game is played only on configurations with empty buffer, except for the initial configuration which might contain some buffer messages. This implies that the TSO program that is described by the game implicitly follows SC semantics.

6 Group II

This group contains TSO games where both players are allowed to update the buffer *only* before their own move. Let player X be the player that has a winning strategy and player Y her opponent. Note that

$$\begin{tikzcd}
\mathsf{c} \arrow[r, "\sigma_X", "\mathsf{instr}_\iota"'] \arrow[d, "\mathsf{up}^*"] \arrow[dr, "\bar{\sigma}_X"] & \mathsf{c}' \arrow[r, "\bar{\sigma}_Y"] \arrow[d, "\mathsf{up}^*"] & \mathsf{c}'' \\
\bar{\mathsf{c}} \arrow[r, "\mathsf{instr}_\iota"'] & \tilde{\mathsf{c}}' \arrow[ur, "\sigma_Y"'] &
\end{tikzcd}$$

Figure 4: Commutative diagram of strategies in games of group II, in the case where $\mathsf{instr}_\iota \neq \mathtt{rd}(\mathsf{x},\mathsf{d})$.

this differs from the previous section, in which the players X and Y were defined based on their updating capabilties.

Similar to the argumentation for Group I, we want to show that player X also has a winning strategy where she empties the buffer in each move. But, in contrast to before, this time there is an exception: Since the player has to update the buffer *before* her move, by updating a memory variable she might disable a read transition that she intended to execute. Thus, we do not require her to empty the buffer in that case.

Formally, let $\mathcal{G} = \langle \mathsf{C}, \mathsf{C}_X, \mathsf{C}_Y, \rightarrow, \mathsf{C}_F \rangle$ be a TSO game where both players are allowed to perform buffer updates exactly before their own moves. Suppose σ_X is a winning strategy for player X and some configuration c_0. We construct another strategy $\bar{\sigma}_X$ for player X. Let $\mathsf{c} \in \mathsf{C}_X$, $\mathsf{c}' = \sigma_X(\mathsf{c})$ and $\bar{\mathsf{c}}$ as in the previous section, i.e. the unique configuration such that $\mathsf{c} \xrightarrow{\mathsf{up}^*}_{\mathcal{P}} \bar{\mathsf{c}}$ and the buffers of c are empty. Suppose that $\mathsf{c} \xrightarrow{\mathsf{instr}_\iota}_{\mathcal{P}} \mathsf{c}'$, where instr_ι is not a read instruction. Then, starting from c, updating all buffer messages does not change that the transition from $\mathcal{S}(\mathsf{c})(\iota)$ to $\mathcal{S}(\mathsf{c}')(\iota)$ is enabled. Thus, instr_ι can also be executed from $\bar{\mathsf{c}}$. We call the resulting configuration $\tilde{\mathsf{c}}'$ and observe that $\bar{\mathsf{c}} \rightarrow_{\mathcal{P}} \tilde{\mathsf{c}}'$ and $\mathsf{c}' \xrightarrow{\mathsf{up}^*} \tilde{\mathsf{c}}'$. We define $\bar{\sigma}_X(\mathsf{c}) = \tilde{\mathsf{c}}'$. This can be seen in Figure 4. Note that $\tilde{\mathsf{c}}'$ may have at most one message in its buffers. In the other case, where there is no transition from c to c' other than read instructions, we define $\bar{\sigma}_X(\mathsf{c}) = \sigma_X(\mathsf{c}) = \mathsf{c}'$.

Claim 9. *$\bar{\sigma}_X$ is a winning strategy for* c_0.

Proof. First, suppose that $\mathsf{c}_0 \in \mathsf{C}_X$ and let σ_Y be an arbitrary strategy of player Y. We define another (non-positional) strategy $\bar{\sigma}_Y$, that depends on the last two configurations, by $\bar{\sigma}_Y(\mathsf{c},\mathsf{c}') = \sigma_Y(\bar{\sigma}_X(\mathsf{c}))$. We observe that for all $\mathsf{c} \in \mathsf{C}_X$, it holds that $\bar{\sigma}_Y(\mathsf{c}, \sigma_X(\mathsf{c})) = \sigma_Y(\bar{\sigma}_X(\mathsf{c}))$. It follows that the play P_1 induced by σ_X and $\bar{\sigma}_Y$ and the play P_2 induced by $\bar{\sigma}_X$ and σ_Y agree on every second configuration, i.e. the configurations in C_X. In particular, the sequence of visited global TSO configurations is the same in both plays. Since σ_X is winning, it means that P_1 is winning for player X and thus also P_2 is winning. Because we chose σ_Y arbitrarily, it follows that $\bar{\sigma}_X$ is a winning strategy.

Otherwise, if $\mathsf{c}_0 \in \mathsf{C}_Y$, we consider the successors of c_0 instead. We note that $\bar{\sigma}_X$ must also be a winning strategy for each $\mathsf{c} \in \mathrm{Post}(\mathsf{c}_0)$. But then, we can apply the previous arguments to each of those configurations and conclude that $\bar{\sigma}_X$ is a winning strategy for all of them. Thus, it is also a winning strategy for c_0. □

We conclude that if player X has a winning strategy σ_X, then she also has a winning strategy $\bar{\sigma}_X$ where she empties the buffers before every turn in which she does not perform a read operation. By symmetry, the same holds true for player Y. Thus, we can limit our analysis to this type of strategies. We see that the number of messages in the buffers is bounded: Suppose that the game is in configuration

$\mathsf{c} \in \mathsf{C}_X$. Then, $\bar{\sigma}_X$ either empties the buffer and adds at most one new message, or it performs a transition due to a read instruction, which does not increase the size of the buffers. The analogous argumentation holds for player Y. Hence, we can reduce the game to a game on bounded buffers, which is finite state and thus decidable.

Given the configuration c_0 as above, we construct a finite game $\mathcal{G}' = \langle \mathsf{C}', \mathsf{C}'_X, \mathsf{C}'_Y, \rightarrow', \mathsf{C}'_F \rangle$ as follows. The set C'_X contains all configurations from C_X which have at most as many buffer messages than c_0 (or at most one message, if c_0 has empty buffers): $\mathsf{C}'_X = \{\mathsf{c} \in \mathsf{C}_X \mid |\mathcal{B}(\mathsf{c})| \leq \max\{1, |\mathcal{B}(\mathsf{c}_0)|\}\}$, where $|\mathcal{B}| = \sum_{\iota \in \mathcal{I}} |\mathcal{B}(\iota)|$. The set C'_Y is defined accordingly. Note that both sets are finite. Lastly, $\rightarrow'$ is defined as the restriction of $\rightarrow$ to configurations of $\mathcal{G}'$, and $\mathsf{C}'_F = \mathsf{C}_F \cap \mathsf{C}'_A$. We define $\bar{\sigma}'_X$ to be the restriction of $\bar{\sigma}_X$ to C'_X. Since $\bar{\sigma}'_X(\mathsf{c}) \in \mathsf{C}'_Y$ for all $\mathsf{c} \in \mathsf{C}'_X$, $\bar{\sigma}'_X$ is indeed a valid strategy for $\mathcal{G}'$. In particular, it is the restriction of $\bar{\sigma}_X$ to $\mathcal{G}'$.

Claim 10. *$\bar{\sigma}'_X$ is a winning strategy for c_0 in $\mathcal{G}'$.*

Proof. First, consider the case where $\mathsf{c}_0 \in \mathsf{C}_X$. Let σ'_Y be a strategy for player Y in $\mathcal{G}'$ and let σ_Y be an arbitrary extension of σ'_Y to $\mathcal{G}$. The play P induced by $\bar{\sigma}_X$ and σ_Y in $\mathcal{G}$ is the same as the play P' induced by $\bar{\sigma}'_X$ and σ'_Y in $\mathcal{G}'$. Since $\bar{\sigma}_X$ is a winning strategy, P is a winning play. It follows that P' must also be a winning strategy. Since σ'_Y was arbitrary, it follows that $\bar{\sigma}'_X$ is a winning strategy and c_0 is winning in $\mathcal{G}'$. ☐

Theorem 11. *The safety problem for games of group II is* ExpTime*-complete.*

Proof. By Claim 9 and Claim 10, if a configuration c_0 is winning for player A in game $\mathcal{G}$, then it is also winning in $\mathcal{G}'$. The same holds true for player B. Thus, the safety problem for $\mathcal{G}$ is equivalent to the safety problem for $\mathcal{G}'$. Similar to the games of group I, $\mathcal{G}'$ is finite and has exponentially many configurations. By Lemma 2 and Theorem 3, we can again conclude that the safety problem is ExpTime-complete. ☐

7 Group III

This group consists of all games where exactly one player has control over the buffer updates, and additionally the game where both players are allowed to update buffer messages *after* their own move. Intuitively, all of them have in common that the TSO program can attribute a buffer update to one specific player. If only one player can update messages, this is clear. In the other game, the first player who observes that a buffer message has reached the memory is not the one who has performed the buffer update. Thus, the program is able to punish misbehaviour, i.e. not following protocols or losing messages.

We will show that the safety problem is undecidable for this group of games. To accomplish that, we reduce the state reachability problem of PCS to the safety problem of each game. Since the former problem is undecidable, so is the latter.

The case where player A is allowed to perform buffer updates at any time is called the *A-TSO game*. It is explained in detail in the following. The other cases work similar, but require slightly different program constructions. They are presented in the appendix [28].

Consider the A-TSO game, i.e. the case where player A can update messages at any time, but player B can never do so. Given a PCS $\mathcal{L} = \langle \mathsf{S}, \mathsf{M}, \rightarrow_{\mathcal{L}} \rangle$ and a final state $\mathsf{s}_F \in \mathsf{S}$, we construct a TSO program $\mathcal{P}$ that simulates $\mathcal{L}$. We design the program such that s_F is reachable in $\mathcal{L}$ if and only if player B wins the safety game induced by $\mathcal{P}$. Thus, the construction gives her the initiative to decide which transitions of $\mathcal{L}$ will be simulated. Meanwhile, the task of player A is to take care of the buffer updates.

$\mathcal{P}$ consists of three processes Proc^1, Proc^2 and Proc^3, that operate on the variables $\{\mathtt{x_{wr}}, \mathtt{x_{rd}}, \mathtt{y}\}$ over the domain $\mathsf{M} \uplus \{0, 1, \perp\}$. The first process simulates the control flow and the message channel of the PCS $\mathcal{L}$. The second process provides a mean to read from the channel. The only task of the third process is to prevent deadlocks, or rather to make any deadlocked player lose. Proc^3 achieves this with four states: the initial state, an intermediate state, and one winning state for each player, respectively. If one of the players cannot move in both Proc^1 and Proc^2, they have to take a transition in Proc^3. From the initial state of this process, there exists only one outgoing transition, which is to the intermediate state. From there, the other player can move to her respective winning state and the process will only self-loop from then on. For player A, her state is winning because she can refuse to update any messages, which will ensure that player B keeps being deadlocked in Proc^1 and Proc^2. For player B, her state simply is contained in $\mathsf{Q}_F^{\mathcal{P}}$. In the following, we will mostly omit Proc^3 from the analysis and just assume that both players avoid reaching a configuration where they cannot take any transition in either Proc^1 or Proc^2.

As mentioned above, we will construct Proc^1 and Proc^2 to simulate the perfect channel system in a way that gives player B the control about which channel operation will be simulated. To achieve this, each channel operation will need an even number of transitions to be simulated in $\mathcal{P}$. Since player B starts the game, this means that after every fully completed simulation step, it is again her turn and she can initiate another simulation step as she pleases. Furthermore, during the simulation of a skip or send operation, we want to prevent player A from executing Proc^2, since this process is only needed for the receive operation. Suppose that we want to block player A from taking a transition $\mathsf{q} \xrightarrow{\mathsf{instr}}_{\mathcal{P}} \mathsf{q}'$. We add a new transition $\mathsf{q}' \xrightarrow{\mathtt{skip}}_{\mathcal{P}} \mathsf{q}_F$, where $\mathsf{q}_F \in \mathcal{S}_F^{\mathcal{P}}$. Hence, reaching q' is immediately losing for player A, since player B can respond by moving to q_F.

Next, we will describe how Proc^1 and Proc^2 simulate the perfect channel system $\mathcal{L}$. For each transition in $\mathcal{L}$, we construct a sequence of transitions in Proc^1 that simulates both the state change and the channel behaviour of the $\mathcal{L}$-transition. To achieve this, Proc^1 uses its buffer to store the messages of the PCS's channel. In particular, to simulate a send operation $!\mathsf{m}$, Proc^1 adds the message $\langle \mathtt{x_{wr}}, \mathsf{m} \rangle$ to its buffer. For receive operations, Proc^1 cannot read its own oldest buffer message, since it is overshadowed by the more recent messages. Thus, the program uses Proc^2 to read the message from memory and copies it to the variable $\mathtt{x_{rd}}$, where it can be read by Proc^1. We call the combination of reading a message m from $\mathtt{x_{wr}}$ and writing it to $\mathtt{x_{rd}}$ the *rotation* of m.

While this is sufficient to simulate all behaviours of the PCS, it also allows for additional behaviour that is not captured by $\mathcal{L}$. More precisely, we need to ensure that each channel message is received *once and only once*. Equivalently, we need to prevent the *loss* and *duplication* of messages. This can happen due to multiple reasons.

The first phenomenon that allows the loss of messages is the seeming lossiness of the TSO buffer. Although it is not strictly lossy, it can appear so: Consider an execution of $\mathcal{P}$ that simulates two send operations $!\mathsf{m}_1$ and $!\mathsf{m}_2$, i.e. Proc^1 adds $\langle \mathtt{x_{wr}}, \mathsf{m}_1 \rangle$ and $\langle \mathtt{x_{wr}}, \mathsf{m}_2 \rangle$ to its buffer. Assume that player A decides to update both messages to the memory, without Proc^2 performing a message rotation in between. The first message m_1 is overwritten by the second message m_2 and is lost beyond recovery.

To prevent this, we extend the construction of Proc^1 such that it inserts an auxiliary message $\langle \mathtt{y}, 1 \rangle$ into its buffer after the simulation of each send operation. After a message rotation, that is, after Proc^2 copied a message from $\mathtt{x_{wr}}$ to $\mathtt{x_{rd}}$, the process then resets the value of $\mathtt{x_{wr}}$ to its initial value $\perp$. Next, the process checks that $\mathtt{y}$ contains the value 0, which indicates that only one message was updated to the memory. Now, player A is allowed to update exactly one $\langle \mathtt{y}, 1 \rangle$ buffer message, after which Proc^2 resets $\mathtt{y}$ to 0. To ensure that player A has actually updated only one message in this step, Proc^2 then checks that $\mathtt{x_{wr}}$ is still empty. Since player A is exclusively responsible for buffer updates, Proc^2 deadlocks her whenever one of these checks fails.

In the next scenario, we discover a different way of message loss. Consider again an execution of $\mathcal{P}$ that simulates two send operations $!\mathsf{m}_1$ and $!\mathsf{m}_2$. Assume Player A updates m_1 to the memory and Proc^2 performs a message rotation. Immediately afterwards, the same happens to m_2, without Proc^1 simulating a receive operation in between. Again, m_1 is overwritten by m_2 before being received, thus it is lost.

Player A is prevented from losing a message in this way by disallowing her to perform a complete message rotation (including the update of one $\langle \mathsf{y}, 1\rangle$-message and the reset of the variables) entirely on her own. More precisely, we add a winning transition for player B to Proc^2 that she can take if and only if player A is the one initiating the update of $\langle \mathsf{y}, 1\rangle$. On the other hand, player A can prevent player B from performing two rotations right after each other by refusing to update the next buffer message until Proc^1 initiates the simulation of a receive operation.

Lastly, we investigate message duplication. This occurs if Proc^1 simulates two receive operations without Proc^2 performing a message rotation in between. In this case, the most recently rotated message is received twice.

The program prevents this by blocking Proc^1 from progressing after a receive operation until Proc^2 has finished a full rotation. In detail, at the very end of the message rotation and $\langle \mathsf{y}, 1\rangle$-update, Proc^2 reset the value of $\mathsf{x}_{\mathtt{rd}}$ to its initial value $\perp$. After simulating a receive operation, Proc^1 is blocked until it can read this value from memory.

This concludes the mechanisms implemented to ensure that each channel message is received *once and only once*. Thus, we have constructed an A-TSO game that simulates a perfect channel system. We summarise our results in the following theorem. The formal proof can be found in Appendix B [28].

Theorem 12. *The safety problem for the A-TSO game is undecidable.*

8 Group IV

In TSO games where no player is allowed to perform any buffer updates, there is no communication between the processes at all. A read operation of a process Proc^ι on a variable x either reads the initial value from the shared memory, or the value of the last write of Proc^ι on x from the buffer, if such a write operation has happened.

Thus, we are only interested in the transitions that are enabled for each process, but we do not need to care about the actual buffer content. In particular, the information that we need to capture from the buffers and the memory is the values that each process can read from the variables, and whether a process can execute a memory fence instruction or not. Together with the global state of the current configuration, this completely determines the enabled transitions in the system.

We call this concept the *view* of the processes on the concurrent system and define it formally as a tuple $\mathsf{v} = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}\rangle$, where:

- $\mathcal{S} : \mathcal{I} \to \bigcup_{\iota \in \mathcal{I}} \mathsf{Q}^\iota$ is a global state of $\mathcal{P}$.
- $\mathcal{V} : \mathcal{I} \times \mathsf{Vars} \to \mathsf{Dom}$ defines which value each process reads from a variable.
- $\mathcal{F} : \mathcal{I} \to \{\mathsf{true}, \mathsf{false}\}$ represents the possibility to perform a memory fence instruction.

Given a view $\mathsf{v} = \langle \mathcal{S}, \mathcal{V}, \mathcal{F}\rangle$, we write $\mathcal{S}(\mathsf{v})$, $\mathcal{V}(\mathsf{v})$ and $\mathcal{F}(\mathsf{v})$ for the global program state $\mathcal{S}$, the value state $\mathcal{V}$ and the fence state $\mathcal{F}$ of v.

The view of a configuration c is denoted by $\mathsf{v}(\mathsf{c})$ and defined in the following way. First, $\mathcal{S}(\mathsf{v}(\mathsf{c})) = \mathcal{S}(\mathsf{c})$. For all $\iota \in \mathcal{I}$ and $\mathsf{x} \in \mathsf{Vars}$, if $\mathcal{B}(\mathsf{c})(\iota)|_{\{\mathsf{x}\}\times\mathsf{Dom}} = \langle \mathsf{x}, \mathsf{d}\rangle \cdot \mathsf{w}$, then $\mathcal{V}(\mathsf{v}(\mathsf{c}))(\iota, \mathsf{x}) = \mathsf{d}$. Otherwise,

$\mathcal{V}(\mathsf{v}(\mathsf{c}))(\iota,\mathsf{x}) = \mathcal{M}(\mathsf{c})(\mathsf{x})$. Lastly, $\mathcal{F}(\mathsf{v}(\mathsf{c}))(\iota) = \mathsf{true}$ if and only if $\mathcal{B}(\mathsf{c})(\iota) = \varepsilon$. We extend the notation to sets of configurations in the usual way, i.e. $\mathsf{v}(\mathsf{C}') = \{\mathsf{v}(\mathsf{c}) \mid \mathsf{c} \in \mathsf{C}'\}$.

For $\mathsf{c},\mathsf{c}' \in \mathsf{C}_{\mathcal{P}}$, if $\mathsf{v}(\mathsf{c}) = \mathsf{v}(\mathsf{c}')$, then we write $\mathsf{c} \equiv \mathsf{c}'$ and say that c and c' are *view-equivalent*. In such a case, a local process of $\mathcal{P}$ cannot differentiate between c and c' in the sense that the enabled transitions in both configurations are the same. Lemma 13 captures this idea formally.

Lemma 13. *For all* $\mathsf{c}_1,\mathsf{c}_2,\mathsf{c}_3 \in \mathsf{C}_{\mathcal{P}}$*,* $\iota \in \mathcal{I}$ *and* $\mathsf{instr} \in \mathsf{Instrs}$ *with* $\mathsf{c}_1 \xrightarrow{\mathsf{instr}_\iota} \mathsf{c}_2$ *and* $\mathsf{c}_1 \equiv \mathsf{c}_3$*, there exists a* $\mathsf{c}_4 \in \mathsf{C}_{\mathcal{P}}$ *such that* $\mathsf{c}_3 \xrightarrow{\mathsf{instr}_\iota} \mathsf{c}_4$ *and* $\mathsf{c}_2 \equiv \mathsf{c}_4$*.*

Proof. We first show that instr_ι is enabled at c_3. Since $\mathsf{c}_1 \equiv \mathsf{c}_3$, it holds that $\mathcal{S}(\mathsf{c}_1) = \mathcal{S}(\mathsf{c}_3)$. Furthermore, if $\mathsf{instr}_\iota = \mathtt{rd}(\mathsf{x},\mathsf{d})_\iota$, then $\mathcal{V}(\mathsf{v}(\mathsf{c}_1))(\iota,\mathsf{x}) = \mathcal{V}(\mathsf{v}(\mathsf{c}_3))(\iota,\mathsf{x}) = \mathsf{d}$. Also, if $\mathsf{instr}_\iota = \mathtt{mf}_\iota$, then $\mathcal{F}(\mathsf{v}(\mathsf{c}_1))(\iota) = \mathcal{F}(\mathsf{v}(\mathsf{c}_3))(\iota) = \varepsilon$. From these considerations and the definition of the TSO semantics (see Figure 2), it follows that instr_ι is indeed enabled at c_3.

Let c_4 be the configuration obtained after performing instr_ι, i.e. $\mathsf{c}_3 \xrightarrow{\mathtt{rd}(\mathsf{x},\mathsf{d})_\iota} \mathsf{c}_4$. It holds that $\mathcal{S}(\mathsf{c}_4) = \mathcal{S}(\mathsf{c}_2) = \mathcal{S}(\mathsf{c}_1)[\iota \leftarrow \mathcal{S}(\mathsf{c}_2)(\iota)]$. If $\mathsf{instr}_\iota = \mathtt{wr}(\mathsf{x},\mathsf{d})_\iota$, then $\mathcal{V}(\mathsf{v}(\mathsf{c}_4)) = \mathcal{V}(\mathsf{v}(\mathsf{c}_2)) = \mathcal{V}(\mathsf{v}(\mathsf{c}_1))[(\iota,\mathsf{x}) \leftarrow \mathsf{d}]$ and $\mathcal{F}(\mathsf{v}(\mathsf{c}_4)) = \mathcal{F}(\mathsf{v}(\mathsf{c}_2)) = \mathcal{F}(\mathsf{v}(\mathsf{c}_1))[\iota \leftarrow \mathsf{false}]$. Otherwise, $\mathcal{V}(\mathsf{v}(\mathsf{c}_4)) = \mathcal{V}(\mathsf{v}(\mathsf{c}_2)) = \mathcal{V}(\mathsf{v}(\mathsf{c}_1))$ and $\mathcal{F}(\mathsf{v}(\mathsf{c}_4)) = \mathcal{F}(\mathsf{v}(\mathsf{c}_2)) = \mathcal{F}(\mathsf{v}(\mathsf{c}_1))$. In all cases it follows that $\mathsf{c}_2 \equiv \mathsf{c}_4$. ☐

We define a finite safety game played on TSO views and show that we can restrict our analysis to this game. Let $\mathcal{G} = \langle \mathsf{C},\mathsf{C}_A,\mathsf{C}_B,\rightarrow,\mathsf{C}_F\rangle$ be a TSO game where neither player can perform any updates. We define a new game $\mathcal{G}' = \langle \mathsf{V},\mathsf{V}_A,\mathsf{V}_B,\rightarrow',\mathsf{V}_F\rangle$ that is played on the views of $\mathcal{G}$. We define $\mathsf{V}_A = \{\mathsf{v}(\mathsf{c})_A \mid \mathsf{c}_A \in \mathsf{C}_A\}$, $\mathsf{V}_B = \{\mathsf{v}(\mathsf{c})_B \mid \mathsf{c}_B \in \mathsf{C}_B\}$, $\mathsf{V} = \mathsf{V}_A \cup \mathsf{V}_B$ and $\mathsf{V}_F = \{\mathsf{v}(\mathsf{c})_A \mid \mathsf{c}_A \in \mathsf{C}_F\}$. Lastly, $\mathsf{v}(\mathsf{c}) \rightarrow' \mathsf{v}(\mathsf{c}')$ if and only if $\mathsf{c} \rightarrow \mathsf{c}'$. This is well-defined by Lemma 13.

Lemma 14. *A configuration* $\mathsf{c}_0 \in \mathsf{C}$ *is winning (for player A / B) in* $\mathcal{G}$ *if and only if the view* $\mathsf{v}_0 = \mathsf{v}(\mathsf{c}_0) \in \mathsf{V}$ *is winning (for player A / B) in* $\mathcal{G}'$*.*

Proof. To simplify notation, we extend $\mathsf{v}(\mathsf{c})$ to configurations of TSO games by $\mathsf{v}(\mathsf{c}_A) = \mathsf{v}(\mathsf{c})_A$ and $\mathsf{v}(\mathsf{c}_B) = \mathsf{v}(\mathsf{c})_B$ for $\mathsf{c}_A \in \mathsf{C}_A$ and $\mathsf{c}_B \in \mathsf{C}_B$. Hence, we can write $\mathsf{V}_A = \mathsf{v}(\mathsf{C}_A)$ and similar.

Suppose c_0 is winning for some player X with (positional) strategy σ_X and consider the case $\mathsf{c}_0 \in \mathsf{C}_X$. In the following, we will define a (non-positional) strategy σ'_X for $\mathcal{G}'$.

First, we need an auxiliary function $f : \mathsf{C} \times \mathsf{V} \rightarrow \mathsf{C}$ that fulfills the condition: For all $\mathsf{c} \in \mathsf{C}$ and $\mathsf{v} \in \mathsf{V}$ such that $\mathsf{v}(\mathsf{c}) \rightarrow' \mathsf{v}$, it holds that $\mathsf{c} \rightarrow f(\mathsf{c},\mathsf{v})$ and $\mathsf{v} = \mathsf{v}(f(\mathsf{c},\mathsf{v}))$. Intuitively, f selects a successor of c with view v. Such a function exists by Lemma 13.

For n even and a sequence $\mathsf{v}_0,\ldots,\mathsf{v}_n$, iteratively define $\mathsf{c}_{2i-1} = \sigma(\mathsf{c}_{2i-2})$ and $\mathsf{c}_{2i} = f(\mathsf{c}_{2i-1},\mathsf{v}_{2i})$ for $i = 1,\ldots,n/2$. Then, $\sigma'_X(\mathsf{v}_0,\ldots,\mathsf{v}_n) = \mathsf{v}(\sigma_X(\mathsf{c}_n))$. We will show that σ'_X is a winning strategy for v_0. Consider a positional strategy σ'_Y for player Y in $\mathcal{G}'$. We define a positional strategy σ_Y for player Y in $\mathcal{G}$ by $\sigma_Y(\mathsf{c}) = f(\mathsf{c},\sigma'_Y(\mathsf{v}(\mathsf{c})))$. Consider the play $\mathsf{P} = \mathsf{c}_0,\mathsf{c}_1,\ldots$ induced by σ_X and σ_Y, and the play $\mathsf{P}' = \mathsf{v}_0,\mathsf{v}_1,\ldots$ induced by σ'_X and σ'_Y.

We proof by induction over k, that (i) $\mathsf{v}_k = \mathsf{v}(\mathsf{c}_k)$ and (ii) c_k of P coincides with c_k as in the definition of σ'_X. In this context, we refer to the latter with $\bar{\mathsf{c}}_k$. For $k = 0$, $\mathsf{v}_0 = \mathsf{v}(\mathsf{c}_0)$ and $\mathsf{c}_0 = \bar{\mathsf{c}}_0$ by definition. For k odd, $\mathsf{c}_k = \sigma_X(\mathsf{c}_{k-1}) = \bar{\mathsf{c}}_k$ by the induction hypothesis. Also,

$$\mathsf{v}_k = \sigma'_X(\mathsf{v}_0,\ldots,\mathsf{v}_{k-1}) = \mathsf{v}(\sigma_X(\bar{\mathsf{c}}_{k-1})) = \mathsf{v}(\sigma_X(\mathsf{c}_{k-1})) = \mathsf{v}(\mathsf{c}_k)\ .$$

For $k > 0$ even,

$$\mathsf{c}_k = \sigma_Y(\mathsf{c}_{k-1}) = f(\mathsf{c}_{k-1},\sigma'_Y(\mathsf{v}(\mathsf{c}_{k-1}))) = f(\bar{\mathsf{c}}_{k-1},\sigma'_Y(\mathsf{v}_{k-1})) = f(\bar{\mathsf{c}}_{k-1},\mathsf{v}_k) = \bar{\mathsf{c}}_k\ .$$

Lastly,

$$\mathsf{v}(\mathsf{c}_k) = \mathsf{v}(\sigma_Y(\mathsf{c}_{k-1})) = \mathsf{v}(f(\mathsf{c}_{k-1}, \sigma'_Y(\mathsf{v}(\mathsf{c}_{k-1})))) = \mathsf{v}(f(\mathsf{c}_{k-1}, \sigma'_Y(\mathsf{v}_{k-1}))) = \mathsf{v}(f(\mathsf{c}_{k-1}, \mathsf{v}_k)) = \mathsf{v}_k ,$$

where the last equality follows from the definition of f.

Since σ_X is a winning strategy for c_0, P is a winning play for player X. From the definition of V_F it follows that P' is a winning play in $\mathcal{G}'$ and thus v_0 is winning for player X. Note that by Lemma 1, we could have chosen a positional strategy in place of σ'_X. Since we did not put any restrictions on the identity of player X, this concludes both the *if* and the *only if* direction of the proof for the case $\mathsf{c}_0 \in \mathsf{C}_X$.

Otherwise, if $\mathsf{c}_0 \in \mathsf{C}_Y$, we consider all configurations of $\text{Post}(\mathsf{c}_0)$ instead. We have the following chain of equivalences: c_0 is winning $\iff$ all $\mathsf{c} \in \text{Post}(\mathsf{c}_0)$ are winning $\iff$ all $\mathsf{v} \in \mathsf{v}(\text{Post}(\mathsf{c}_0))$ are winning $\iff$ all $\mathsf{v} \in \text{Post}(\mathsf{v}(\mathsf{c}_0))$ are winning $\iff$ $\mathsf{v}(\mathsf{c}_0)$ is winning. Here, the second equivalence applies the first case of this proof and the third equivalence uses $\text{Post}(\mathsf{v}(\mathsf{c}_0)) = \mathsf{v}(\text{Post}(\mathsf{c}_0))$, which follows from the definition of $\mathcal{G}'$. ☐

Theorem 15. *The safety problem for games in group IV is* EXPTIME*-complete.*

Proof. By Lemma 14, the safety problem for $\mathcal{G}$ is equivalent to the safety problem of $\mathcal{G}'$, which is played on views. Since there exist only exponentially many views, EXPTIME-completeness follows from Lemma 2 and Theorem 3, similar to Group I and II. ☐

9 Conclusion and Future Work

In this work we have addressed for the first time the game problem for programs running under weak memory models in general and TSO in particular. Surprisingly, our results show that depending on when the updates take place, the problem can turn out to be undecidable or decidable. In fact, there is a subtle difference between the decidable (group I, II and IV) and undecidable (group III) TSO games. For the former games, when a player is taking a turn, the system does not know who was responsible for the last update. But for the latter games, the last update can be attributed to a specific player. Another surprising finding is the complexity of the game problem for the groups I, II and IV which is EXPTIME-complete in contrast with the non-primitive recursive complexity of the reachability problem for programs running under TSO and the undecidability of the repeated reachability problem.

In future work, the games where exactly one player has control over the buffer seem to be the most natural ones to expand on. In particular, the A-TSO game (where player A can update before and after her move) and the B-TSO game (same, but for player B). On the other hand, the games of groups I, II and IV seem to be degenerate cases and therefore rather uninteresting. In particular, they do not seem to be more powerful than games on programs that follow SC semantics.

Another direction for future work is considering other memory models, such as the partial store ordering semantics, the release-acquire semantics, and the ARM semantics. It is also interesting to define stochastic games for programs running under TSO as extension of the probabilistic TSO semantics [1].

References

[1] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Raj Aryan Agarwal, Adwait Godbole & Shankara Narayanan Krishna (2022): *Probabilistic Total Store Ordering*. In Ilya Sergey, editor: *Programming Languages and Systems - 31st European Symposium on Programming, ESOP 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*, *Lecture Notes in Computer Science* 13240, Springer, pp. 317–345, doi:10.1007/978-3-030-99336-8_12.

[2] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani & Tuan Phong Ngo (2016): *The Benefits of Duality in Verifying Concurrent Programs under TSO*. In Josée Desharnais & Radha Jagadeesan, editors: *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, *LIPIcs* 59, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 5:1–5:15, doi:10.4230/LIPIcs.CONCUR.2016.5.

[3] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani & Tuan Phong Ngo (2018): *A Load-Buffer Semantics for Total Store Ordering*. *Log. Methods Comput. Sci.* 14(1), doi:10.23638/LMCS-14(1:9)2018.

[4] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Carl Leonardsson & Ahmed Rezine (2012): *Counter-Example Guided Fence Insertion under TSO*. In Cormac Flanagan & Barbara König, editors: *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, *Lecture Notes in Computer Science* 7214, Springer, pp. 204–219, doi:10.1007/978-3-642-28756-5_15.

[5] Parosh Aziz Abdulla, Mohamed Faouzi Atig, Florian Furbach, Adwait Amit Godbole, Yacoub G. Hendi, Shankara Narayanan Krishna & Stephan Spengler (2023): *Parameterized Verification under TSO with Data Types*. In Sriram Sankaranarayanan & Natasha Sharygina, editors: *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I*, *Lecture Notes in Computer Science* 13993, Springer, pp. 588–606, doi:10.1007/978-3-031-30823-9_30.

[6] Parosh Aziz Abdulla, Mohamed Faouzi Atig & Ngo Tuan Phong (2015): *The Best of Both Worlds: Trading Efficiency and Optimality in Fence Insertion for TSO*. In Jan Vitek, editor: *Programming Languages and Systems - 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, *Lecture Notes in Computer Science* 9032, Springer, pp. 308–332, doi:10.1007/978-3-662-46669-8_13.

[7] Parosh Aziz Abdulla, Mohamed Faouzi Atig & Rojin Rezvan (2020): *Parameterized verification under TSO is PSPACE-complete*. *Proc. ACM Program. Lang.* 4(POPL), pp. 26:1–26:29, doi:10.1145/3371094.

[8] Parosh Aziz Abdulla, Ahmed Bouajjani & Julien d'Orso (2008): *Monotonic and Downward Closed Games*. *J. Log. Comput.* 18(1), pp. 153–169, doi:10.1093/logcom/exm062.

[9] Parosh Aziz Abdulla & Bengt Jonsson (1994): *Undecidable Verification Problems for Programs with Unreliable Channels*. In Serge Abiteboul & Eli Shamir, editors: *Automata, Languages and Programming, 21st International Colloquium, ICALP94, Jerusalem, Israel, July 11-14, 1994, Proceedings*, *Lecture Notes in Computer Science* 820, Springer, pp. 316–327, doi:10.1007/3-540-58201-0_78.

[10] Parosh Aziz Abdulla & Bengt Jonsson (1996): *Verifying Programs with Unreliable Channels*. *Inf. Comput.* 127(2), pp. 91–101, doi:10.1006/inco.1996.0053.

[11] ARM (2014): *ARM Architecture Reference Manual, ARMv7-A and ARMv7-R edition*. Available at https://developer.arm.com/documentation/ddi0406/latest/.

[12] Mohamed Faouzi Atig (2020): *What is decidable under the TSO memory model?* *ACM SIGLOG News* 7(4), pp. 4–19, doi:10.1145/3458593.3458595.

[13] Mohamed Faouzi Atig, Ahmed Bouajjani, Sebastian Burckhardt & Madanlal Musuvathi (2010): *On the verification problem for weak memory models*. In Manuel V. Hermenegildo & Jens Palsberg, editors: *Proceedings*

of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010, ACM, pp. 7–18, doi:10.1145/1706299.1706303.

[14] Ahmed Bouajjani, Egor Derevenetc & Roland Meyer (2013): *Checking and Enforcing Robustness against TSO*. In Matthias Felleisen & Philippa Gardner, editors: *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, *Lecture Notes in Computer Science* 7792, Springer, pp. 533–553, doi:10.1007/978-3-642-37036-6_29.

[15] Ahmed Bouajjani, Roland Meyer & Eike Möhlmann (2011): *Deciding Robustness against Total Store Ordering*. In Luca Aceto, Monika Henzinger & Jirí Sgall, editors: *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, *Lecture Notes in Computer Science* 6756, Springer, pp. 428–440, doi:10.1007/978-3-642-22012-8_34.

[16] Daniel Brand & Pitro Zafiropulo (1983): *On Communicating Finite-State Machines*. *J. ACM* 30(2), pp. 323–342, doi:10.1145/322374.322380.

[17] Ashok K. Chandra, Dexter Kozen & Larry J. Stockmeyer (1981): *Alternation*. *J. ACM* 28(1), pp. 114–133, doi:10.1145/322234.322243.

[18] Alain Finkel & Philippe Schnoebelen (2001): *Well-structured transition systems everywhere!* *Theor. Comput. Sci.* 256(1-2), pp. 63–92, doi:10.1016/S0304-3975(00)00102-X.

[19] Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors (2002): *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*. *Lecture Notes in Computer Science* 2500, Springer, doi:10.1007/3-540-36387-4.

[20] IBM (2021): *Power ISA, Version 3.1b*. Available at https://files.openpower.foundation/s/dAYSdGzTfW4j2r2/download/OPF_PowerISA_v3.1B.pdf.

[21] Intel Corporation (2012): *Intel 64 and IA-32 Architectures Software Developers Manual*. Available at https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

[22] Leslie Lamport (1979): *How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs*. *IEEE Trans. Computers* 28(9), pp. 690–691, doi:10.1109/TC.1979.1675439.

[23] René Mazala (2001): *Infinite Games*. In Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors: *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, *Lecture Notes in Computer Science* 2500, Springer, pp. 23–42, doi:10.1007/3-540-36387-4_2.

[24] Scott Owens, Susmit Sarkar & Peter Sewell (2009): *A Better x86 Memory Model: x86-TSO*. In Stefan Berghofer, Tobias Nipkow, Christian Urban & Makarius Wenzel, editors: *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, *Lecture Notes in Computer Science* 5674, Springer, pp. 391–407, doi:10.1007/978-3-642-03359-9_27.

[25] Philippe Schnoebelen (2002): *Verifying lossy channel systems has nonprimitive recursive complexity*. *Inf. Process. Lett.* 83(5), pp. 251–261, doi:10.1016/S0020-0190(01)00337-4.

[26] Peter Sewell, Susmit Sarkar, Scott Owens, Francesco Zappa Nardelli & Magnus O. Myreen (2010): *x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors*. *Commun. ACM* 53(7), pp. 89–97, doi:10.1145/1785414.1785443.

[27] SPARC International, Inc. (1994): *SPARC Architecture Manual Version 9*. Available at https://sparc.org/wp-content/uploads/2014/01/SPARCV9.pdf.gz.

[28] Stephan Spengler & Sanchari Sil (2023): *TSO Games – On the decidability of safety games under the total store order semantics*, doi:10.48550/arXiv.2309.02862.

CGAAL: Distributed On-The-Fly ATL Model Checker with Heuristics[*]

Falke B. Ø. Carlsen
Department of Computer Science,
Aalborg University, Denmark
falkeboc@cs.aau.dk

Lars Bo P. Frydenskov
Department of Computer Science,
Aalborg University, Denmark
larsbopark@gmail.com

Nicolaj Ø. Jensen
Department of Computer Science,
Aalborg University, Denmark
noje@cs.aau.dk

Jener Rasmussen
jener@jener.dk

Mathias M. Sørensen
mathiasmehlsoerensen@gmail.com

Asger G. Weirsøe
asger@weircon.dk

Mathias C. Jensen
Department of Computer Science,
Aalborg University, Denmark
mcje@cs.aau.dk

Kim G. Larsen
Department of Computer Science,
Aalborg University, Denmark
kgl@cs.aau.dk

We present CGAAL, our efficient on-the-fly model checker for alternating-time temporal logic (ATL) on concurrent game structures (CGS). We present how our tool encodes ATL as extended dependency graphs with negation edges and employs the distributed on-the-fly algorithm by Dalsgaard et al. Our tool offers multiple novel search strategies for the algorithm, including DHS which is inspired by PageRank and uses the in-degree of configurations as a heuristic, IHS which estimates instability of assignment values, and LPS which estimates the distance to a state satisfying the constituent property using linear programming. CGS are input using our modelling language LCGS, where composition and synchronisation are easily described. We prove the correctness of our encoding, and our experiments show that our tool CGAAL is often one to three orders of magnitude faster than the popular tool PRISM-games on case studies from PRISM's documentation and among case studies we have developed. In our evaluation, we also compare and evaluate our search strategies, and find that our custom search strategies are often significantly faster than the usual breadth-first and depth-first search strategies.

1 Introduction

Software plays a large role in our everyday lives, making decisions, enabling efficient communication, ensuring safety, and many more critical tasks. Furthermore, the complexity of the software and the decisions they have to make are ever-increasing due the to interconnectivity and reactive nature of modern systems. These software systems mutually depend on, communicate with, and guide each other based on their collective and internal states. Even a few interconnected systems that are not necessarily themselves too complex can give rise to an extremely complex system as a whole. Safety-critical software that operates in contexts where errors could lead to human casualties or significant capital losses requires methods to verify their correctness. Unfortunately, such methods are difficult to implement due to the sheer state-space explosion that arises from the inherent parallel composition of systems.

In this paper, we consider systems that can be expressed as discrete multiplayer games in which the actors can perform actions concurrently. That is, in each configuration of the game, each player simultaneously chooses an action they wish to perform and the resulting decision vector then deterministically

[*]Funded by the VILLUM INVESTIGATOR project S4OS.

results in the next configuration of the game. Such concurrent games are extremely expressive and can be used to model a variety of complex systems and lends themselves inherently, by their concurrent nature, to describing multi-actor systems. The properties that we wish to verify on these systems are those that can be expressed by Alternating-time Temporal Logic (ATL) [1], a game theoretic extension of Computation Tree Logic (CTL) [2, 3], that like CTL can be used to describe safety and liveness properties. ATL extends CTL by replacing the existential and universal path quantifiers with so-called coalition quantifiers. These coalition quantifiers describe properties of the game in which a coalition of players can either enforce some desired outcome or cannot avoid an outcome. Of course, proof of satisfaction of such property is done by finding a strategy showing that the players can indeed enforce said property or by showing that no matter what strategy the players follow they cannot avoid said property.

One existing tool that can model check concurrent (stochastic) games is PRISM-games [12], an extension of the probabilistic model checker PRISM [11]. PRISM excels at stochastic models such as discrete- and continuous-time Markov chains, Markov decision processes, and (priced) probabilistic timed automata, for which it can verify various properties described in LTL , CSL, and probabilistic CTL*. There are multiple verification engines in PRISM, both symbolic and explicit-state. The PRISM-games extension focuses on stochastic games in which game-theoretic approaches are needed. PRISM-games can among other things synthesise strategies and reason about equilibria-based properties, where players may have distinct, but not necessarily conflicting objectives. These properties are typically described with probabilistic ATL with rewards. In version 3.0 [12], PRISM-games added support for concurrent stochastic games.

In [14], Liu and Smolka present a global and a local algorithm to compute fixed-point boolean vertex assignments in dependency graphs with directed hyper-edges representing dependencies between vertex values. The global algorithm has a better worst-case running time of the two, but the on-the-fly local algorithm only explores the graph as needed to compute the fixed-point assignment. Many problems have since been encoded as dependency graphs. Notably, model-checking problems for CTL can be encoded as dependency graphs and the algorithms by Liu and Smolka can be used to compute the satisfaction relation enabling model-checking using dependency graphs. J. F. Jensen et al. [9] show that weighted CTL can be encoded in dependency graphs as well and introduce symbolic edges to handle weights efficiently. In [10], M. C. Jensen et al. introduce symbolic dependency graphs, an encoding of probabilistic weighted CTL, and an implementation of a local and global algorithm. A. Dalsgaard et al. [4] present a distributed extension to the local algorithm to boost the performance and show how to model Petri Nets problems using CTL and dependency graphs. Furthermore, they extend dependency graphs by adding components and negation edges to solve formulae with negations. An abstract dependency graph framework is presented by S. Enevoldsen et al. in [6] where they generalise the various extensions of the dependency graphs with a general algorithm for any Noetherian partial order domain. In the closely related work of [5], S. Enevoldsen et at. uses abstract dependency graphs to model check probabilistic ATL for weighted stochastic games. In multiple works [4, 10] the local algorithm is found to be faster more often than the global algorithm in practice.

In this paper, we introduce the tool CGAAL (Concurrent Games AALborg), a model checker implemented in Rust that allows for the verification of ATL properties on concurrent games. CGAAL works by encoding the model checking problem as an extended dependency graph and by implementing the on-the-fly local distributed algorithm by Dalsgaard et al. [4]. The concurrent games are described and inputted to the tool using LCGS, our PRISM-language inspired model language, and for specific ATL formulae, we can output a strategy describing how to ensure satisfaction given a positive result. Furthermore, CGAAL implements a variety of strategies for searching the problem state space: a heuristic inspired by PageRank [15] and two heuristic search strategies that exploit the state vector representation of our model, alongside the usual breadth-first and depth-first approach. We conduct experiments comparing our different search strategies with each other on several case studies and show that having more available compute threads often leads to speed-ups of 1-2 orders of magnitude. We also perform experiments comparing our tool on a selection of case studies to the state-of-the-art PRISM-games. Though while PRISM-games is a tool specialised for probabilistic games we find that our implementation often outperforms PRISM-games and especially whenever we are not required to compute the entire fixed point in which case we are often 1-3 orders of magnitude faster. Again, we emphasise that PRISM-games is meant for probabilistic games and as such this is not a completely fair one-to-one comparison.

Outline. This paper is structured as follows. Section 2 introduces the formal definitions of concurrent games and alternating-time temporal logic. In Section 3 we present how model checking of ATL properties in concurrent games can be encoded in extended dependency graphs. We also give a short explanation of the CERTAINZERO algorithm and our search strategies. In Section 4 we give a short example of how to use CGAAL and the LCGS language. An evaluation of our tool and a comparison with PRISM can be found in Section 5, and we conclude on our findings in Section 6.

2 Definitions

We recall the definitions of concurrent games and alternating-time temporal logic.

Concurrent Games A *concurrent game structure* (CGS) is a tuple $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$ where:

- $k \geq 1$ is a natural number of players. We identify the players with the numbers $1, \dots, k$.
- Q is a finite set of states.
- Π is a finite set of atomic propositions, also called labels.
- A set $\pi(q) \subseteq \Pi$ of propositions are true at $q \in Q$. The function π is called the *labelling function*.
- For each player $a \in \{1, \dots, k\}$ and each state $q \in Q$, a natural number $d_a(q) \geq 1$ of moves are available to player a at state q. The moves of player a at state q are identified with the numbers $1, \dots, d_a(q)$. A move vector at $q \in Q$ is a tuple $v = \langle j_1, \dots, j_k \rangle$ such that $1 \leq j_a \leq d_a(q)$ for each player $a \in \{1, \dots, k\}$. Additionally, given a state $q \in Q$, we write $D(q)$ for the set $\{1, \dots, d_1(q)\} \times \dots \times \{1, \dots, d_k(q)\}$ of all move vectors possible at q. The function D is called the *move function*.
- For each state $q \in Q$ and each move vector $v = \langle j_1, \dots, j_k \rangle \in D(q)$, we have that $\delta(q, v) \in Q$ is the resulting state when each player $a \in \{1, \dots, k\}$ chooses move j_a in the state q. The function δ is called the *transition function*.

A state q' is a *successor* to q if and only if there exists a transition from q to q', i.e. there exists a move vector $v \in D(q)$ such that $\delta(q, v) = q'$. A *computation* is an infinite sequence $\lambda = q_0, q_1, q_2, \dots$ of states, such that for all $i \geq 0$, the state q_{i+1} is a successor of q_i. Given a computation λ and a position $i \geq 0$, we use the following notations:

$\lambda[i]$	The ith state of the computation of λ
$\lambda[0, i]$	The finite prefix $q_0, q_1, \dots, q_i$ of λ
$\lambda[i, \infty]$	The infinite suffix $q_i, q_{i+1}, q_{i+2}, \dots$ of λ

A computation starting in the state q is called a q-computation.

Game strategies Consider a concurrent game structure $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$ over the set $\Sigma = \{1, \ldots, k\}$ of players. A *strategy* for a player $a \in \Sigma$ is a function $z_a : Q \to \mathbb{N}$ that maps every state $q \in Q$ to a natural number, such that $z_a(q) \leq d_a(q)$. In other words, the strategy z_a describes how player a chooses their move in each state. A strategy z_a for a player $a \in \Sigma$ induces a set of computations that the player a can enforce. Given a state $q \in Q$, a set $A \subseteq \Sigma$ of players, and a set $Z_A = \{z_a\}_{a \in A}$ of strategies, one strategy for each player in A, we define the outcomes of the strategies Z_A from q, denoted $out(q, Z_A)$, to be the set of q-computations that the players of A enforce following the strategies of Z_A. That is, a computation $\lambda = q_0, q_1, q_2, \ldots$ is in $out(q, Z_A)$ if $q_0 = q$ and for all positions $i \geq 0$, there is a move vector $v = \langle j_1, \ldots, j_k \rangle \in D$ such that

1. $j_a = z_a(q_i)$ for all players $a \in A$, and
2. $\delta(q_i, v) = q_{i+1}$.

The set $\mathcal{Z}^A = \{\{z_a\}_{a \in A} \mid z_a$ is a strategy for the player $a\}$ contains all sets of strategies, that contain one strategy for each player in $A \subseteq \Sigma$.

Alternating-time Temporal Logic The *alternating-time temporal logic* (ATL) [1] is defined with respect to a finite set Π of *propositions* and a finite set $\Sigma = \{1, ..., k\}$ of players. An ATL formula is given by the abstract syntax:

$$\phi ::= p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid \langle\langle A \rangle\rangle \bigcirc \phi \mid \langle\langle A \rangle\rangle (\phi_1 \mathcal{U} \phi_2) \mid [\![A]\!] (\phi_1 \mathcal{U} \phi_2)$$

where $p \in \Pi$ is a proposition and $A \subseteq \Sigma$ is a set of players. The operators $\langle\langle \cdot \rangle\rangle$ and $[\![\cdot]\!]$ are *path quantifiers*. We will refer to them as "enforce" and "despite", respectively. The $\bigcirc$ ("next") and $\mathcal{U}$ ("until") are *temporal operators*. Additional temporal operators like $\Diamond$ ("eventually") and $\Box$ ("invariant") are derived as usual.

Given a state q of a game structure S, we write $q \vDash \phi$ to indicate that the state q satisfies the property described by ϕ. The satisfactory relation $\vDash$ is defined inductively:

- $q \vDash p$ iff $p \in \pi(q)$.
- $q \vDash \neg\phi$ iff $q \nvDash \phi$.
- $q \vDash \phi_1 \vee \phi_2$ iff $q \vDash \phi_1$ or $q \vDash \phi_2$.
- $q \vDash \langle\langle A \rangle\rangle \bigcirc \phi$ iff there exists a set $Z_A \in \mathcal{Z}^A$ of strategies, such that for all computations $\lambda \in out(q, Z_A)$, we have $\lambda[1] \vDash \phi$.
- $q \vDash \langle\langle A \rangle\rangle (\phi_1 \mathcal{U} \phi_2)$ iff there exists a set $Z_A \in \mathcal{Z}^A$ of strategies, such that for all computation $\lambda \in out(q, Z_A)$ there exists a position $i \geq 0$, such that $\lambda[i] \vDash \phi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \vDash \phi_1$.
- $q \vDash [\![A]\!] (\phi_1 \mathcal{U} \phi_2)$ iff for all sets $Z_A \in \mathcal{Z}^A$ of strategies, we have that there exists a computation $\lambda \in out(q, Z_A)$, such that there exists a position $i \geq 0$, such that $\lambda[i] \vDash \phi_2$ and for all positions $0 \leq j < i$, we have $\lambda[j] \vDash \phi_1$.

3 Model Checking

In order to check if a CGS satisfies an ATL property, CGAAL encodes the problem as an extended dependency graph and finds a fixed-point assignment describing the satisfaction relation.

Extended Dependency Graphs An *extended dependency graph* (EDG) is a tuple $G = \langle C, E, N \rangle$ where C is a finite set of configurations (vertices), $E \subseteq C \times \mathcal{P}(C)$ is a finite set of hyper-edges, and $N \subseteq C \times C$ is a finite set of negation edges.

For a hyper-edge $e = \langle c, T \rangle \in E$ we call c the source configuration and $T \subseteq C$ the set of target configurations. Similarly, c is called the source configuration of the negation edge $\langle c, c' \rangle \in N$. We write $c \rightarrow c'$ if there exists an edge $\langle c, T \rangle \in E$ such that $c' \in T$ and $c \dashrightarrow c'$ if $\langle c, c' \rangle \in N$. Furthermore, we write $c \rightsquigarrow c'$ if $c \rightarrow c'$ or $c \dashrightarrow c'$. An EDG $G = \langle C, E, N \rangle$ is negation safe if there are no $c, c' \in C$ such that $c \dashrightarrow c'$ and $c' \rightsquigarrow^* c$. In what follows, we consider only negation-safe EDGs. Let $dist : C \rightarrow \mathbb{N}_0$ be the maximum number of negation edges throughout all paths starting in a configuration $c \in C$, inductively defined as:

$$dist(c) = \max\{dist(c'') + 1 \mid c', c'' \in C \text{ and } c \rightarrow^* c' \dashrightarrow c''\} \tag{1}$$

By convention $\max \emptyset = 0$. We define the $dist(G)$ of an EDG G as $dist(G) = \max_{c \in C}(dist(c))$. A component K_i of an EDG G, where $i \in \mathbb{N}_0$, is a subgraph induced on G by the sets

- $C_i = \{c \in C \mid dist(c) \leq i\}$
- $E_i = \{\langle c, T \rangle \in E \mid dist(c) \leq i\}$
- $N_i = \{\langle c, c' \rangle \in N \mid dist(c) \leq i\}$

denoting the set of configurations, hyper-edges and negation edges respectively in each respective component. By definition, the component K_0 has no negation edges.

Assignments An assignment $\alpha : C \rightarrow \{0, 1\}$ is a function that assigns boolean values to configurations of an EDG $G = \langle C, E, N \rangle$, where 0 and 1 represent false and true, respectively. We define α_0 as the zero assignment where $\alpha_0(c) = 0$ for all $c \in C$. We assume a component-wise ordering $\sqsubseteq$ on assignments such that we have $\alpha \sqsubseteq \alpha'$ whenever $\alpha(c) \leq \alpha'(c)$ for all $c \in C$. The set of all assignments of an EDG G is denoted by $\mathcal{A}^G$ and $\langle \mathcal{A}^G, \sqsubseteq \rangle$ is a complete lattice.

By Knaster and Tarski's theorem, we can find a minimum fixed point on the complete lattice for any monotonic function. The minimum fixed-point assignment α_{min} of an EDG G, denoted as $\alpha^G_{\text{min}} = \alpha^{K_{dist(G)}}_{\text{min}}$ is defined inductively on the components $K_0, K_1, \ldots, K_{dist(G)}$ of G. For all $0 \leq i \leq dist(G)$, we define $\alpha^{K_i}_{\text{min}}$ to be the minimum fixed-point assignment of the monotonic function $F_i : \mathcal{A}^{C_i} \rightarrow \mathcal{A}^{C_i}$ where

$$F_i(\alpha)(c) = \alpha(c) \vee \left[\bigvee_{\langle c,T \rangle \in E_i} \bigwedge_{c' \in T} \alpha(c') \right] \vee \left[\bigvee_{\langle c,c' \rangle \in N_i} \neg \alpha^{K_{i-1}}_{\text{min}}(c') \right] \tag{2}$$

By convention, the conjunction of $\emptyset$ is true and the disjunction of $\emptyset$ is false. In the component K_0 there are no negation edges, which means the last clause in the disjunction is false for K_0. We will refer to the repeated use of the F_i as the *global algorithm* because it requires building and iterating over the entire EDG. By Equation (2) each configuration in the EDG is comparable to a boolean formula in disjunctive normal form where each clause is another configuration or a negation of another configuration. Clearly, this makes EDGs an expressive structure and we shall now encode ATL model checking in an EDG.

3.1 Encoding of ATL in an EDG

We first establish some definitions related to subsets of move vectors induced by a coalition of players.

Partial Moves. Given a CGS S with the set of states Q and k players, we use $V_1 \times \cdots \times V_k = \mathcal{V} \subseteq D(q)$ where $q \in Q$ to denote *a partial* move in q, where zero or more of the players' moves are fixed, i.e. their set of possible moves contain a single number. For instance, if $\mathcal{V} = \{2\} \times \{1,2,3\}$ in a 2-player game, then player 1 has chosen move 2 while player 2 is still free to choose from 1, 2, or 3. Furthermore, we use the notation $\mathcal{V}[a \mapsto j]$ for a variant $\mathcal{V}'$ of a partial move $\mathcal{V} = V_1 \times \cdots \times V_k$ where player a chooses move j. More precisely

$$\mathcal{V}[a \mapsto j] = V_1' \times \cdots \times V_k' \quad \text{where } 1 \le i \le k \text{ and } V_i' = \begin{cases} \{j\} & \text{if } i = a \\ V_i & \text{otherwise} \end{cases} \tag{3}$$

With this notation in mind, we define the function *pmoves*. The function *pmoves* gives the set of all possible partial moves that follow from the players of the set $A \subseteq \Sigma$ making a combination of moves in the state $q \in Q$:

$$pmoves(q,A) = \{D(q)[a \mapsto j_a][b \mapsto j_b]\cdots \mid \{a,b,\dots\} = A \text{ and } \forall i \in A. 1 \le j_i \le d_i(q)\} \tag{4}$$

For instance, if $D(q) = \{1,2\} \times \{1,2\}$ in a 2-player game and $A = \{1\}$. Then $pmoves(q,A) = \{\{1\} \times \{1,2\}, \{2\} \times \{1,2\}\}$ contains two partial moves. One where player 1 chooses action 1, and another where player 1 chooses action 2. In other words, the *pmoves* function constructs the partial moves that the set A of players can choose from when *working together* in the state q. If A chooses $\mathcal{V} \in pmoves(q,A)$, then $\mathcal{V}$ contains all move vectors that result from the remaining players $\Sigma \backslash A$ also making a choice.

We define a *partial transition function* Δ that produces a set of possible successor states given a state $q \in Q$ and a partial move $\mathcal{V}$. That is

$$\Delta(q,\mathcal{V}) = \{s \mid v \in \mathcal{V} \text{ and } \delta(q,v) = s\} \quad \text{where } \mathcal{V} \subseteq D(q) \tag{5}$$

With EDGs, assignments, and partial moves introduced, we can now define how the satisfaction relation $\models$ can be encoded as an EDG.

Encoding. Given a CGS S, a state q of S, and an ATL state formula ϕ we now construct an EDG where every configuration is either a pair of a state and a formula, or a triple of a state, partial move, and a formula. The triples represent partially evaluated states, where some players' moves are already set. Starting from the initial pair $\langle q, \phi \rangle$, the dependency graph is constructed according to Figures 1 to 6. The figures show which outgoing edges each configuration has and the target configurations of those edges.

Theorem 3.1. *Given a CGS $S = \langle k, Q, \Pi, \pi, d, \delta \rangle$, a state $q \in Q$, and an ATL formula ϕ, let G be the negation-safe EDG rooted in $\langle q, \phi \rangle$ constructed following Figures 1 to 6. Then $\alpha^G_{\min}(\langle q, \phi \rangle) = 1$ if and only if $q \models \phi$.*

The proof of Theorem 3.1 can be found in the extended version of this paper.

Certain Zero In addition to the global algorithm (repeated application of F_i from Equation (2)), CGAAL also implements a local algorithm heavily inspired by the distributed CERTAINZERO algorithm by Dalsgaard et al. [4]. A local algorithm specialises in finding the assignment $\alpha^G_{\min}(\langle q, \phi \rangle)$ of a specific configuration $\langle q, \phi \rangle$ and not necessarily the assignment of the whole EDG. This allows the algorithm to terminate early in many cases. For the CERTAINZERO algorithm, Dalsgaard et al. also introduce a

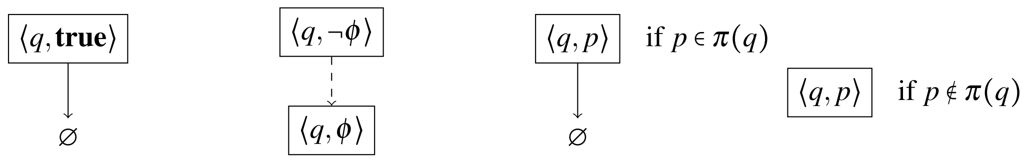

$\langle q, \mathbf{true} \rangle$
$\varnothing$
$\langle q, \neg\phi \rangle$
$\langle q, \phi \rangle$
$\langle q, p \rangle$ if $p \in \pi(q)$
$\varnothing$
$\langle q, p \rangle$ if $p \notin \pi(q)$

Figure 1: EDG encoding of true, negation, and atomic proposition

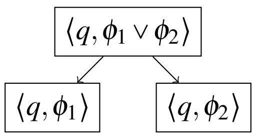

$\langle q, \phi_1 \vee \phi_2 \rangle$
$\langle q, \phi_1 \rangle$
$\langle q, \phi_2 \rangle$

Figure 2: EDG encoding of disjunction

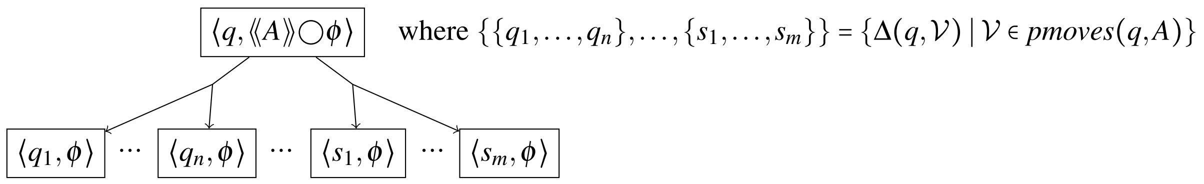

$\langle q, \langle\!\langle A \rangle\!\rangle \bigcirc \phi \rangle$
where $\{\{q_1,\dots,q_n\},\dots,\{s_1,\dots,s_m\}\} = \{\Delta(q,\mathcal{V}) \mid \mathcal{V} \in pmoves(q,A)\}$
$\langle q_1, \phi \rangle$ ⋯ $\langle q_n, \phi \rangle$ ⋯ $\langle s_1, \phi \rangle$ ⋯ $\langle s_m, \phi \rangle$

Figure 3: EDG encoding of next

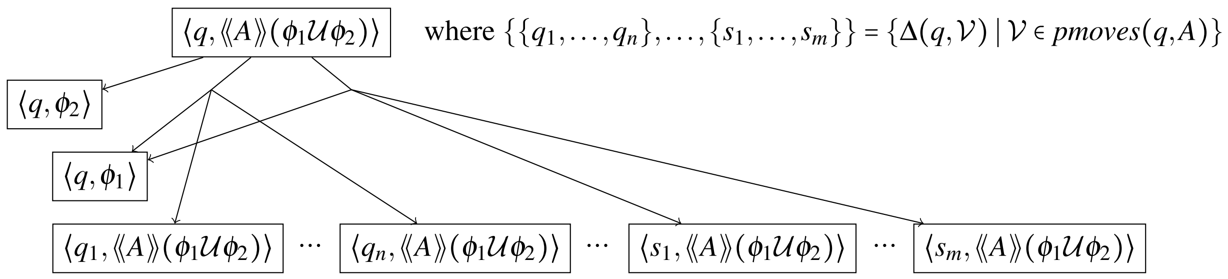

$\langle q, \langle\!\langle A \rangle\!\rangle (\phi_1 \mathcal{U} \phi_2) \rangle$
where $\{\{q_1,\dots,q_n\},\dots,\{s_1,\dots,s_m\}\} = \{\Delta(q,\mathcal{V}) \mid \mathcal{V} \in pmoves(q,A)\}$
$\langle q, \phi_2 \rangle$
$\langle q, \phi_1 \rangle$
$\langle q_1, \langle\!\langle A \rangle\!\rangle (\phi_1 \mathcal{U} \phi_2) \rangle$ ⋯ $\langle q_n, \langle\!\langle A \rangle\!\rangle (\phi_1 \mathcal{U} \phi_2) \rangle$ ⋯ $\langle s_1, \langle\!\langle A \rangle\!\rangle (\phi_1 \mathcal{U} \phi_2) \rangle$ ⋯ $\langle s_m, \langle\!\langle A \rangle\!\rangle (\phi_1 \mathcal{U} \phi_2) \rangle$

Figure 4: EDG encoding of enforce until

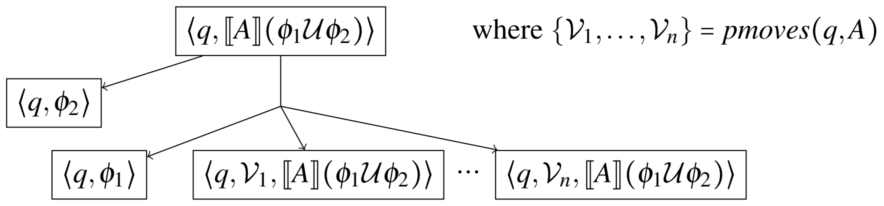

$\langle q, [\![A]\!] (\phi_1 \mathcal{U} \phi_2) \rangle$
where $\{\mathcal{V}_1,\dots,\mathcal{V}_n\} = pmoves(q,A)$
$\langle q, \phi_2 \rangle$
$\langle q, \phi_1 \rangle$
$\langle q, \mathcal{V}_1, [\![A]\!] (\phi_1 \mathcal{U} \phi_2) \rangle$ ⋯ $\langle q, \mathcal{V}_n, [\![A]\!] (\phi_1 \mathcal{U} \phi_2) \rangle$

Figure 5: EDG encoding of despite until

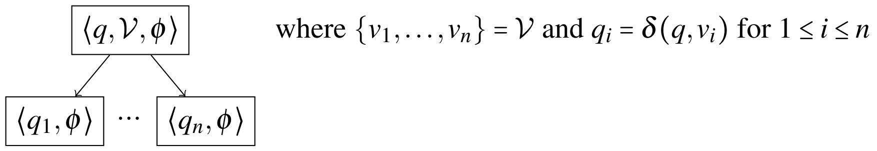

$\langle q, \mathcal{V}, \phi \rangle$
where $\{v_1,\dots,v_n\} = \mathcal{V}$ and $q_i = \delta(q,v_i)$ for $1 \le i \le n$
$\langle q_1, \phi \rangle$ ⋯ $\langle q_n, \phi \rangle$

Figure 6: EDG encoding of partially moved despite until

$$\begin{tikzcd} 0 & & 1 \\ & ? \arrow[ul, dash] \arrow[ur, dash] & \\ & \bot \arrow[u, dash] & \end{tikzcd}$$

Figure 7: Ordering of assignment values for fixed-point computation in the CERTAINZERO algorithm. The top values are more certain.

symbolic assignment with new assignment values. The value $\perp$ indicates that the configuration is unexplored. The value ? (unknown) indicates that the configuration is explored, but its final value has yet to be determined. Lastly, 0 and 1 indicate the final values in the minimum fixed-point assignment α^G_{min}. This induces an ordering as seen in the lattice in Figure 7. During the algorithm, the assignments of the configurations will rise to more and more certain values, and if the assignment of a configuration ever becomes 0 or 1, we can be certain that its final value is 0 or 1, respectively. This is one of the properties that lead to improved performance, since in contrast to the global algorithm where all configurations are initially assigned 0, we can distinguish an initial 0 and a certain 0. Hence, the CERTAINZERO algorithm can terminate early if the root configuration is ever assigned either 0 or 1. Otherwise, it terminates when it can no longer raise any assignment values.

3.2 Search Strategies

The CERTAINZERO algorithm is controlled by a search strategy, which determines in which order the edges are explored and evaluated. Our search strategies for CGAAL include the common breadth-first search (BFS) and depth-first search (DFS) strategies, as well as multiple search strategies based on heuristics. Some of these are discussed in the following subsections. Which strategy is best depends heavily on the shape of the EDG which is determined by the CGS and the ATL formula in question. An evaluation of the strategies can be found in Section 5. The BFS strategy is the default strategy.

Dependency Heuristic Search (DHS) PageRank [15] is an algorithm that was created to estimate the importance of a website based on how many other websites have links to it. The idea has since been used in other areas, such as graph recommendation systems [13] or measuring structural-context similarity with SimRank [7]. Our dependency heuristic search (DHS) uses a similar idea by assuming that configurations with many ingoing edges are important. Finding the assignments of these, results in more back-propagation, bringing us closer to termination. In other words, the heuristic focuses on the *trunk* of the EDG where certain assignments are of high value. Specifically, DHS prioritise edges where the source configuration has a high number of ingoing edges. That is, if e is an edge with source configuration c then:

$$priority(e) = indegree(source(e)) = |\{e' \mid (e' = \langle c', T\rangle \in E \wedge c \in T) \vee e' = \langle c', c\rangle \in N\}| \tag{6}$$

Edges with the same priority are subject to FIFO ordering. However, since we do not know the whole graph in advance, we must continuously update the priority of edges when we explore successors of new configurations. This is only a small overhead with a priority queue data structure.

Linear Programming Heuristic Search (LPS) In this search strategy, we take advantage of how LCGS states can be represented as vectors, i.e. Q is a vector space. Given an edge with source configuration $\langle q, \phi\rangle$, we transform ϕ into a set $\mathcal{L}_\phi$ of linear constraints, each defined as a pair $\langle \mathbf{C}, b\rangle$, where $\mathbf{C}$

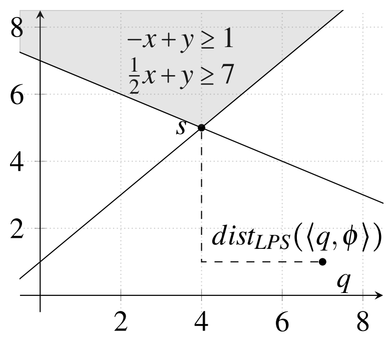


Figure 8: The distance $dist_{LPS}(\langle q,\phi\rangle) = 7$ when $q = \langle 7,1\rangle \in Q = \mathbb{R}^2$ and $\mathcal{L}_\phi$ corresponds to the constraints $-x+y \geq 1$ and $\frac{1}{2}x+y \geq 7$. The state s minimises $||s-q||_1$ while adhering to the constraints.

is a matrix and b is a vector. We prioritise edge e if its source configuration $\langle q,\phi\rangle$ has a low estimated distance to satisfaction as given by:

$$dist_{LPS}(\langle q,\phi\rangle) = \min ||s-q||_1 \quad \text{subject to } \mathbf{C}s \geq b \quad \text{where } s \in Q,\ \langle \mathbf{C},b\rangle \in \mathcal{L}_\phi \tag{7}$$

where $||\cdot||_1$ is the 1-norm, i.e. taxicap distance. The following equation is an equivalent linear programming problem:

$$\min \sum_i x_i \quad \text{subject to } \mathbf{C}s \geq b \text{ and } -x_i \leq s_i - q_i \leq x_i \quad \forall i \quad \text{where } s \in Q,\ \langle \mathbf{C},b\rangle \in \mathcal{L}_\phi \tag{8}$$

A visualisation of the distance $dist_{LPS}$ can be seen on Figure 8, where $Q = \mathbb{R}^2$, $q = \langle 7,1\rangle \in Q$ and

$$\mathcal{L}_\phi = \{\langle \begin{bmatrix} -1 & 1 \\ \frac{1}{2} & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 7 \end{bmatrix} \rangle\}$$

Many methods exist for solving linear programming problems. We use a library called `minilp`[1]. The LPS strategy assumes value-wise close states are structurally close and works best when the constituent variables of states represent non-categorical data. The drawback of this method is the computational overhead of linear programming amplified by the potential need to solve multiple linear programming problems for each configuration based on the structure of ϕ. We reduce some of this overhead by caching the set $\mathcal{L}_\phi$ for each ϕ.

We also support an alternative search strategy involving linear programming. It is called Linear Representative Search (LRS) and it computes the distance described above for the root configuration only. The closest satisfying state s is then saved and edges are prioritised based on the 1-norm distance between s and the state in their source configuration. In other words, we assume that the state s found for the root represents all satisfied states. As a result this search strategy is cheaper than LPS but risks being inaccurate.

Instability Heuristic Search (IHS) The Petri Net model checker TAPAAL [8] implements the CERTAINZERO algorithm as well. Their configurations consist of a marking (a state) and a property, and their default search strategy uses a heuristic that estimates the distance between the marking of the configuration and a marking that satisfies the formula of the configuration. Our novel instability heuristic is inspired by their heuristic, but we differ by acknowledging that since there are negation edges in the EDG, we may not always be looking for a state that satisfies the formula. That is, if the state already

[1]`minilp` crate: `https://crates.io/crates/minilp`

satisfies the associated formula, we estimate the distance to a state that does not instead. This guides the search towards configurations where the assignment is *unstable* and thus may have a high influence. Algorithms 1 and 2 find the $dist_{IHS}$ described above for an edge. The algorithms use an abstract metric *BiDist* which for state-property pairs finds $\langle \hat{t}, \hat{f} \rangle$ where $\hat{t}$ is the distance to the property being true and $\hat{f}$ is the distance to the property being false.

```
Algorithm 1: $dist_{IHS}$
  Input  : An edge $e \in E$
  Output: An estimated distance to a different assignment
1 if $e = \langle c, T \rangle \in E$ then
2     $\langle \hat{t}, \hat{f} \rangle := \bigsqcap_{\langle q,\phi \rangle \in T} BiDist(q, \phi)$;
3     return $\hat{t}$ if $\hat{t} > 0$, otherwise $\hat{f}$
4 else if $e = \langle c, \langle q, \phi \rangle \rangle \in N$ then
5     $\langle \hat{t}, \hat{f} \rangle := BiDist(q, \phi)$;
6     return $\hat{f}$ if $\hat{f} < 0$, otherwise $\hat{t}$
```

```
Algorithm 2: BiDist
  Input  : A state $q$ and formula $\phi$
  Output: A bi-distance $\langle \hat{t}, \hat{f} \rangle$ describing the instability of $\phi$ in $q$
1 if $\phi = expr_1 \lhd expr_2$ and $v = eval(q, expr_1) - eval(q, expr_2)$ then
2     return $\langle v, 0 \rangle$ if $v > 0$, otherwise $\langle 0, v \rangle$
3 else if $\phi = expr_1 \rhd expr_2$ and $v = eval(q, expr_2) - eval(q, expr_1)$ then
4     return $\langle v, 0 \rangle$ if $v > 0$, otherwise $\langle 0, v \rangle$
5 else if $\phi = \neg\phi'$ and $\langle \hat{t}, \hat{f} \rangle = BiDist(q, \phi')$ then
6     return $\langle \hat{f}, \hat{t} \rangle$
7 else if $\phi = \phi_1 \land \phi_2$ then
8     return $BiDist(q, \phi_1) \sqcap BiDist(q, \phi_2)$
9 else if $\phi = \phi_1 \lor \phi_2$ then
10    return $BiDist(q, \phi_1) \sqcup BiDist(q, \phi_2)$
11 else if $\phi = \langle\langle A \rangle\rangle \bigcirc \phi'$ then
12    return $BiDist(q, \phi')$
13 else if $\phi = \langle\langle A \rangle\rangle (\phi_1 \, \mathcal{U} \, \phi_2)$ then
14    return $BiDist(q, \phi_1) \sqcup BiDist(q, \phi_2)$
15 else if $\phi = [\![A]\!] (\phi_1 \, \mathcal{U} \, \phi_2)$ then
16    return $BiDist(q, \phi_1) \sqcup BiDist(q, \phi_2)$
17 // where $\lhd \in \{<, \leq\}, \langle \hat{t}_1, \hat{f}_1 \rangle \sqcap \langle \hat{t}_2, \hat{f}_2 \rangle = \langle \hat{t}_1 + \hat{t}_2, \min\{\hat{f}_1, \hat{f}_2\} \rangle$
18 //       $\rhd \in \{>, \geq\}, \langle \hat{t}_1, \hat{f}_1 \rangle \sqcup \langle \hat{t}_2, \hat{f}_2 \rangle = \langle \min\{\hat{t}_1, \hat{t}_2\}, \hat{f}_1 + \hat{f}_2 \rangle$
```

4 Tool Overview

CGAAL (`https://github.com/d702e20/CGAAL`) is written in Rust and consists of a command-line interface and a verification engine. The primary feature of CGAAL is the verification of ATL properties for CGSs. Verification is either done with a global algorithm or local algorithm, both of which are described in Section 3. If requested, a partial strategy witness can also be computed, which instructs how the given players must play to satisfy the given property. Another feature converts the model into `dot` graph format such that it can be visually rendered with Graphviz.[2]

To model concurrent game structures for CGAAL, we designed a declarative Language for Concurrent Game Structure, called LCGS. In LCGS a CGS is defined with a series of declarations such that it is possible to find the successors of any state from the declarations alone. The language acts as an abstract representation of the CGS and allows us to save memory at the small cost of having to evaluate the expressions of declarations whenever successors are explored. The syntax of the language is inspired by PRISM-lang [11] used by the PRISM model checker to model stochastic multi-player games with rewards. However, LCGS differs in multiple ways. For instance, LCGS has player templates instead of modules, and templates have no effect unless there exists an instance of it. Additionally, synchronisations affecting the internal state of a player are much easier to declare.

Example Use As an example, we want to check if a cowboy can guarantee to stay alive in a three-way Mexican standoff. The standoff is simulated in rounds and in each round a cowboy can choose to wait, shoot the cowboy to the right, or shoot the cowboy to the left. If a cowboy is hit by two bullets, he dies.

```
1  const max_health = 2;
2
3  template cowboy
4
5      // How many bullets a cowboy can be hit by before dying
6      health : [0 .. max_health] init max_health;
7      health' = max(health - opp_right.shoot_left - opp_left.shoot_right, 0);
8
9      // A proposition used by ATL formulae
10     label alive = health > 0;
11
12     // The actions available to each cowboy
13     [wait] 1;
14     [shoot_right] health > 0 && opp_right.health > 0;
15     [shoot_left] health > 0 && opp_left.health > 0;
16
17 endtemplate
18
19 // The three cowboys in the Mexican standoff
20 player billy = cowboy [opp_right=clayton, opp_left=jesse];
21 player clayton = cowboy [opp_right=jesse, opp_left=billy];
22 player jesse = cowboy [opp_right=billy, opp_left=clayton];
```

Listing 1: LCGS implementation of a Mexican standoff

We model this scenario in Listing 1. The cowboy template is declared on lines 3-17. Herein, on line 6, we define that each cowboy can be hit by two bullets before being incapacitated, but we make this number a constant on line 1 so it is easy to change. Each cowboy has their own health variable and the combination of the values of these variables makes up a state. Line 7 defines how the health of a cowboy is updated in each transition. The value of `opp_right.shoot_left` and `opp_left.shoot_right` are 1 if this cowboy was shot by the given opponent to the right or left, respectively, otherwise 0. On line 10

[2]Graphviz: `https://graphviz.org/`

we define a label called alive, which is a proposition that is true if the cowboy has more than zero health. Lines 13-15 define the actions of the cowboy template, and the expression to the right of the name is a condition defining in which states the action is available. As can be seen, a cowboy can always wait, but only shoot if they and their target are alive. Lastly, we declare three instances of the cowboy template on lines 20-22. We define the right and left opponent in the relabelling function after the name of the template. Any identifier in a template can be relabelled to another expression as long as the result is syntactically and semantically correct.

The ATL formula $\langle\!\langle$`billy`$\rangle\!\rangle\Box$`billy.alive` is satisfied if the cowboy `billy` has a strategy to stay alive. We can now run CGAAL with the following command: `./cgaal solver -m standoff.lcgs -f billy-can-stay-alive.atl`, and CGAAL will tell us that the property is not satisfied. Billy has no strategy that can guarantee his survival.

5 Evaluation

To evaluate our tool, we run several experiments. In our experiments we compare our global algorithm, our local algorithm using our various search strategies, and the established tool PRISM-games. We use several different concurrent game case studies. Some of these are PRISM-games case studies adapted to LCGS for CGAAL and determinised if needed. Others are well-known algorithms and games constructed during the development of CGAAL. We make the PRISM-lang and LCGS implementations of models as identical as possible, such that the state spaces are comparable. Some of the concurrent games and related ATL formulae used in the experiments are presented below. Queries marked with $\top$ (resp. $\bot$) are satisfied (resp. not satisfied), and queries marked with † may terminate early as we are not required to compute the entire fixed-point:

- **Mexican-Standoff**: In this model N cowboys stand in a circle, each with a revolver. At each moment they can choose to shoot another cowboy or do nothing. If a cowboy is hit by B bullets, they are incapacitated. We run the following queries:

$$MS_N^B\phi_1^{\bot\dagger} = \langle\!\langle p_1\rangle\!\rangle\Box p_1\_alive$$
$$MS_N^B\phi_2^{\bot} = \langle\!\langle p_1\rangle\!\rangle\Diamond\neg p_1\_alive$$
$$MS_N^B\phi_3^{\top} = \langle\!\langle\{p_i\in\Sigma \mid 1\equiv i \mod 2\}\rangle\!\rangle\Box p_1\_alive$$

- **Gossiping Girls**: In this model, N nodes each know a piece of information. The nodes can exchange information by calling each other. Our experiments include both a fully connected network of nodes and a circular network of nodes. We run the following queries:

$$GG_N\phi_1^{\top\dagger} = \langle\!\langle\Sigma\rangle\!\rangle(calls\leq 10\,\mathcal{U}\,all\_girls\_know\_all)$$
$$GG_N\phi_2^{\top\dagger} = \langle\!\langle\Sigma\rangle\!\rangle(calls\leq 10\,\mathcal{U}\,only\_p_1\_knows\_all)$$
$$GG_N\phi_3^{\top\dagger} = \langle\!\langle\Sigma\rangle\!\rangle(calls\leq 10\,\mathcal{U}\,p_1\_knows\_all)$$
$$GG_N\phi_4^{\top\dagger} = \langle\!\langle\Sigma\rangle\!\rangle\Diamond calls > 10\wedge all\_girls\_know\_all$$
$$GG_N\phi_5^{\top\dagger} = \langle\!\langle\varnothing\rangle\!\rangle\Diamond calls > 10$$
$$GG_N\phi_6^{\bot\dagger} = \langle\!\langle p_1\rangle\!\rangle(calls\leq 10\,\mathcal{U}\,p_1\_knows\_all)$$
$$GG_N\phi_7^{\top\dagger} = \langle\!\langle\Sigma\setminus\{p_1\}\rangle\!\rangle\Diamond everyone\_except\_p_1\_knows\_all$$

- **Robot Coordination**: In this model, four robots move orthogonally on a N by N grid. Each robot needs to reach the opposite corner of where they start, but crashing into each other is fatal. We run the following queries:

$$\begin{aligned}
RC_N\phi_1^{\perp} &= \langle\langle p_1, p_2, p_3\rangle\rangle \lozenge p_1\_at\_target \land p_2\_at\_target \\
RC_N\phi_2^{\top} &= all\_robots\_at\_home \\
RC_N\phi_3^{\top\dagger} &= \langle\langle p_1, p_2, p_3, p_4\rangle\rangle \lozenge all\_robots\_at\_targets \\
RC_N\phi_4^{\perp} &= \langle\langle p_1, p_3, p_4\rangle\rangle \lozenge p_1\_at\_target \land \neg p_1\_crashed
\end{aligned}$$

Each experiment is run with a time limit of two hours, allocated 128 GB of memory, and has 32 cores available regardless of how many worker threads CGAAL may spawn. All experiments are run on several identical AMD EPYC 7642 based servers, allowing only one experiment per node at a time to reduce noise in the results.

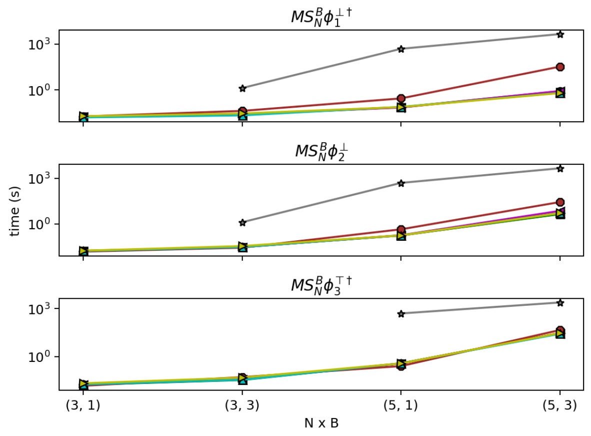
Results Here we provide a select set of experimental results that roughly exemplify our findings in general. For a more thorough set of results along with all of our experimental data and the Python script processing it, see the git repository https://github.com/d702e20/evaluation-results/.

We find that the local on-the-fly algorithm is often one order of magnitude faster than the global algorithm regardless of the search strategy. However, the global algorithm can compete with and sometimes outperforms the local algorithm in cases where the local algorithm cannot terminate early. This matches observations made by A. Dalsgaard et al. [4] and M. C. Jensen et al. [10].

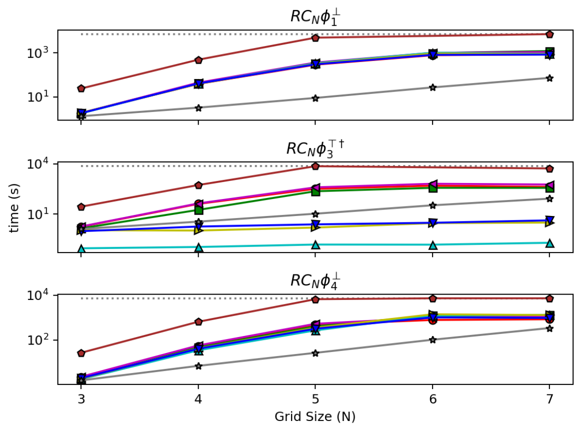
PRISM-games is a stochastic model checker and checks queries involving both probability and rewards, capabilities which are irrelevant in our test cases. Therefore, in many cases, CGAAL's local algorithm is two orders of magnitude faster than PRISM-games, typically when the model has many synchronisations that affect internal states of modules such as in Mexican Standoff and Gossiping Girls as seen in Figures 9a and 9c. Such synchronisations also require a high amount of PRISM-lang code, while being easily expressed in LCGS. However, there are a few cases where PRISM-games is faster than CGAAL, e.g. $RC_N\phi_1^{\perp}$ and $RC_N\phi_4^{\perp}$ as seen in Figure 9b. Here the local algorithm cannot terminate early.

The choice of search strategy sometimes has a discernible impact on the performance of the local algorithm. Which search strategy is best varies from case to case and no definitive best strategy. The BFS strategy does not work well when the local algorithm can terminate early based on information found multiple moves into the model, but its low overhead is beneficial when early termination is not possible. The IHS strategy is often a good choice when early termination is possible. The LPS strategy is often significantly slower than the others, which is unsurprising given its overhead of solving linear programming problems. In models where differences in the states correspond to a distance in space, such as in robot coordination, the LPS strategy performs notably better and is even faster than both BFS and DFS on $RC_4\phi_3^{\top\dagger}$. The lightweight version, LRS, which only solves the linear problem once, matches LPS in these cases but is also generally better due to its lower overhead.

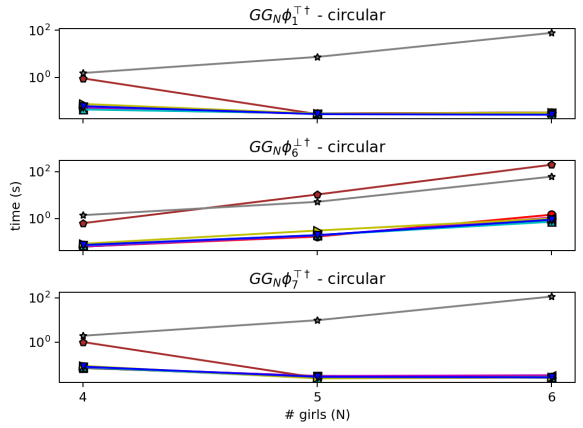
A general trend in our experiments is that we see an increase in the execution speed of our distributed implementation as we increase the number of compute threads available. The only times we do not see a speed-up with an increased number of compute threads are when the models are small enough such that the overhead of managing the additional threads is significant. Similarly, a single thread performs better than a few threads in many cases, since there is no communication overhead with only one compute thread. In general, depending on the search strategy employed, we see an improvement on a scale of one to two orders of magnitude when increasing the number of compute threads as exemplified in Figure 9d.



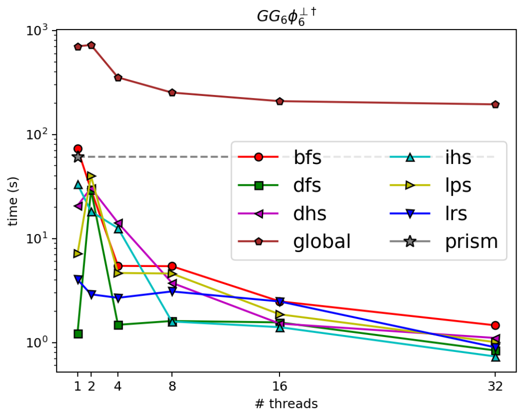
(a) Mexican-standoff: horizontal axis is (N,B) tuples with N being the number of cowboys in the model and B the number of bullets the cowboys can be hit by.



(b) Robot Coordination: horizontal axis is the size of the grid the robots are to manoeuvre on.



(c) Gossiping girls in a circular topology: horizontal axis being the number of girls in the circle.



(d) Gossiping girls with six girls in a circular topology: horizontal axis being the number of compute threads used in our distributed algorithm. Note that PRISM was always run with just one thread.

Figure 9: Select experimental results. CGAAL-results in Figures 9a to 9c show the best result for each search strategy when varying between 1 and 32 threads.

6 Conclusion

In this paper, we present CGAAL, our model checker of alternating-time temporal logic properties in concurrent games. CGAAL checks such properties by encoding the problem as an extended dependency graph and then computes the satisfaction relation using the distributed local on-the-fly CERTAINZERO algorithm by Dalsgaard et al. [4]. We provide multiple novel search strategies for the algorithm and allow concurrent games to be expressed in our language LCGS. Our experiments show that the local on-the-fly algorithm outperforms the global algorithm in the majority of cases. We also find that CGAAL outperforms the state-of-the-art tool PRISM-games by being up to two orders of magnitude faster, especially in models where synchronisations affect the internal state of modules and whenever we are not required to compute the entire fixed point. However, this comparison is unfair, since our test only uses a fraction of PRISM-games feature set.

CGAAL is still in early development and much work is needed before it competes with PRISM in terms of capabilities. However, dependency graphs have been used for encoding various model-checking problems, and we intend to incorporate these techniques into CGAAL.

References

[1] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (2002): *Alternating-Time Temporal Logic*. *J. ACM* 49(5), p. 672–713, doi:10.1145/585265.585270.

[2] Edmund M. Clarke & E. Allen Emerson (1982): *Design and synthesis of synchronization skeletons using branching time temporal logic*. In Dexter Kozen, editor: *Logics of Programs*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 52–71, doi:10.1007/BFb0025774.

[3] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, Roderick Bloem et al. (2018): *Handbook of model checking*. 10, Springer, doi:10.1007/978-3-319-10575-8.

[4] Dalsgaard, Andreas E. and Enevoldsen, Søren and Fogh, Peter and Jensen, Lasse S. and Jepsen, Tobias S. and Kaufmann, Isabella and Larsen, Kim G. and Nielsen, Søren M. and Olesen, Mads Chr. and Pastva, Samuel and Srba, Jiří (2017): *Extended Dependency Graphs and Efficient Distributed Fixed-Point Computation*. In Wil van der Aalst & Eike Best, editors: *Application and Theory of Petri Nets and Concurrency*, Springer International Publishing, Cham, pp. 139–158, doi:10.1007/978-3-319-57861-3_10.

[5] Søren Enevoldsen, Mathias Claus Jensen, Kim Guldstrand Larsen, Anders Mariegaard & Jiří Srba (2020): *Verification of Multiplayer Stochastic Games via Abstract Dependency Graphs*. *LOPSTR2020*, doi:10.1007/978-3-030-68446-4_13.

[6] Søren Enevoldsen, Kim Guldstrand Larsen & Jirí Srba (2022): *Extended abstract dependency graphs*. *Int. J. Softw. Tools Technol. Transf.* 24(1), pp. 49–65, doi:10.1007/s10009-021-00638-8.

[7] Glen Jeh & Jennifer Widom (2002): *SimRank: a measure of structural-context similarity*. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, ACM, pp. 538–543, doi:10.1145/775047.775126.

[8] Jonas F. Jensen, Thomas Nielsen, Lars K. Oestergaard & Jiří Srba (2016): *TAPAAL and Reachability Analysis of P/T Nets*, pp. 307–318. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-662-53401-4_16.

[9] Jonas Finnemann Jensen, Kim Guldstrand Larsen, Jiri Srba & Lars Kaerlund Østergaard (2016): *Efficient model-checking of weighted CTL with upper-bound constraints*. *International Journal on Software Tools for Technology Transfer* 18(4), pp. 409–426, doi:10.1007/s10009-014-0359-5.

[10] Mathias Claus Jensen, Anders Mariegaard & Kim Guldstrand Larsen (2019): *Symbolic model checking of weighted PCTL using dependency graphs*. *NASA Formal Methods Symposium*, pp. 298–315, doi:10.1007/978-3-030-20652-9_20.

[11] M. Kwiatkowska, G. Norman & D. Parker (2011): *PRISM 4.0: Verification of Probabilistic Real-time Systems*. In G. Gopalakrishnan & S. Qadeer, editors: *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, *LNCS* 6806, Springer, pp. 585–591, doi:10.1007/978-3-642-22110-1_47.

[12] M. Kwiatkowska, G. Norman, D. Parker & G. Santos (2020): *PRISM-games 3.0: Stochastic Game Verification with Concurrency, Equilibria and Time*. In: *Proc. 32nd International Conference on Computer Aided Verification (CAV'20)*, *LNCS* 12225, Springer, pp. 475–487, doi:10.1007/978-3-030-53291-8_25.

[13] Sangkeun Lee, Minsuk Kahng & Sang goo Lee (2015): *Constructing compact and effective graphs for recommender systems via node and edge aggregations*. *Expert Systems with Applications* 42(7), pp. 3396–3409, doi:10.1016/j.eswa.2014.11.062.

[14] Xinxin Liu & Scott A. Smolka (1998): *Simple Linear-Time Algorithms for Minimal Fixed Points (Extended Abstract)*. In Kim Guldstrand Larsen, Sven Skyum & Glynn Winskel, editors: *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, *Lecture Notes in Computer Science* 1443, Springer, pp. 53–66, doi:10.1007/BFb0055040.

[15] Lawrence Page, Sergey Brin, Rajeev Motwani & Terry Winograd (1998): *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report, Stanford Digital Library Technologies Project. Available at `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.1768`.

(Un)Decidability Bounds of the Synthesis Problem for Petri Games

Paul Hannibal
University of Oldenburg,
Lower Saxony, Germany
paul.jonathan.hannibal1@uni-oldenburg.de

Petri games are a multi-player game model for the automatic synthesis of distributed systems, where the players are represented as tokens on a Petri net and are grouped into environment players and system players. As long as the players move in independent parts of the net, they do not know of each other; when they synchronize at a joint transition, each player gets informed of the entire causal history of the other players.

We show that the synthesis problem for two-player Petri games under a global safety condition is NP-complete and it can be solved within a non-deterministic exponential upper bound in the case of up to 4 players. Furthermore, we show the undecidability of the synthesis problem for Petri games with at least 6 players under a local safety condition.

1 Introduction

A Petri game is a model for distributed, reactive systems. It is played on a Petri net where each place is either a system place or an environment place. The tokens on system places are system players and they control which transitions to take next. The tokens on environment places are uncontrollable environment players. Essential for Petri games is the informedness of the players. As long as the players move in independent parts of the net, they do not know of each other; when taking a joint transition they exchange information about their complete causal history.

A winning strategy of the system players reacts to all options of the environment players while satisfying a winning condition. Thereby, a decision of a system player is based on its causal history, which grows infinitely in a Petri net with loops. Different causal histories allow different decisions. The *synthesis problem* asks whether there is a winning strategy of the system players. There have been several positive results on deciding the synthesis problem for Petri games, obtained by restricting the number of players [8, 7] or restricting the concurrent behaviour [13]. Also, an approach to bounded synthesis has been proposed [6]. These papers considered as winning conditions either local safety conditions where some 'bad' places must be avoided or global safety conditions requiring that some sets of places are considered as 'bad' markings that must not be reached simultaneously.

Petri games are related to the model of control games played by multiple processes on Zielonka automata. These games are also based on the causal memory of their processes. A control game is a composition of local processes. The processes communicate via shared actions that are either controllable or uncontrollable. A strategy consists of one local controller for each process that can restrict controllable actions based on the causal past of the process. As in Petri games, a strategy must take into account all uncontrollable behaviour in order to win. Unlike Petri games, one of the most common winning conditions is a local termination condition. A formal relationship of the two models has been presented in [1], where translations from Petri games into control games and back have been presented such that there is a weak bisimulation between the winning strategies of the two models. When translating a control game into a Petri game, the processes are turned one-by-one into players. These players switch between system and environment players. The winning condition stays the same. The number of places (or states) blows up exponentially when a game is translated in either direction.

Decidability results for control games have been obtained by restricting the communication architecture [17, 10] or restricting the concurrent behaviour [15, 16]. Another class of decidable control games are decomposable games [11] that come with a local termination condition. Decomposable games are decidable with up to 4 players [11]. In Sec. 3, we show that the synthesis problem for Petri games under a global safety condition with up to 4 players is decidable within an exponential upper time bound, and for two-player Petri games this problem is NP-complete.

In [12], it has been shown that the synthesis problem is undecidable for control games with at least 6 processes under a local termination condition. This result, together with the translation into Petri games in [1], implies that there are 6-player Petri games that are equipped with a local termination condition for which the synthesis problem is undecidable. In Sec. 4, we show in a direct proof that the synthesis problem for Petri games with 6 players is also undecidable under a local safety condition.

The synthesis problem is of great interest because it automates the error-prone implementation process while delivering implementations that are correct by construction. It was first introduced in [3]. Pnueli and Rosner introduced a setting of synchronous processes that communicate via shared variables [19]. For a single process, this setting is known to be decidable [2, 18]. For multiple processes, the setting of Pnueli and Rosner is known to be undecidable [19]. There have been positive decidability results on specific architectures with multiple processes, including pipelines [20], rings [14], and acyclic architectures [9]. However, all the positive results for multiple processes have non-elementary complexity.

2 Foundations

In this section, we define branching processes and unfoldings as in [4]. Also, we define Petri games and their winning strategies as in [8].

Some notation: the *power set* of a set A is denoted by $2^A = \{B \mid B \subseteq A\}$, the *set of nonempty finite subsets* of A by $2^A_{nf} = \{B \mid B \subseteq A \wedge B \neq \emptyset \wedge B \text{ is finite}\}$, and the *set of finite subsets* of A by 2^A_f.

A *Petri net* or simply *net* is a structure $N = (\mathscr{P}, \mathscr{T}, pre, post, In)$, where $\mathscr{P}$ is the (possibly infinite) set of *places*, $\mathscr{T}$ is the (possibly infinite) set of *transitions*, *pre* and *post* are *flow mappings*, $In \subseteq \mathscr{P}$ is the *initial marking*, and the following properties hold: $\mathscr{P} \cap \mathscr{T} = \emptyset$, $pre : \mathscr{T} \to 2^{\mathscr{P}}_{nf}$, $post : \mathscr{T} \to 2^{\mathscr{P}}_f$. A Petri net is called finite if $\mathscr{P} \cup \mathscr{T}$ is a finite set. The flow mappings *pre* and *post* are extended to places as usual: $\forall p \in \mathscr{P} : pre(p) = \{t \in \mathscr{T} \mid p \in post(t)\}$ and $\forall p \in \mathscr{P} : post(p) = \{t \in \mathscr{T} \mid p \in pre(t)\}$. A *marking* M of a Petri net N is a *multiset* over $\mathscr{P}$. In particular, In is a marking. By convention, a net named N has the components $(\mathscr{P}, \mathscr{T}, pre, post, In)$, and analogously for net with decorated names like N_0, N_1, N_2, where the components are equally decorated.

A transition $t \in \mathscr{T}$ is *enabled* at marking M if $pre(t) \subseteq M$. If t is enabled, the transition t can be *fired*, such that the new marking is $M' = M - pre(t) + post(t)$. This is denoted as $M[t\rangle M'$. This notation is extended to sequences of enabled transitions $M[t_1 \dots t_n\rangle M'$. A marking M is *reachable* if there exists a sequence of successively enabled transitions $(t_k)_{k \in \{1,\dots,n\}}$ and $In[t_1 \dots t_n\rangle M$. This sequence can be empty. The set of all reachable markings of a net N is denoted as $\mathscr{R}(N)$. A Petri net N is called *safe*, if for all reachable markings $M(p) \leq 1$ holds for all $p \in \mathscr{P}$. Then, M is a subset of P. All Petri nets considered in this paper are safe.

A *node* x is a place or a transition $x \in \mathscr{P} \cup \mathscr{T}$. The binary *flow relation* $\mathscr{F}$ on nodes is defined as follows: $x\mathscr{F}y$ if $x \in pre(y)$. A node x is a *causal predecessor* of y, denoted as $x \leq y$, if $x\mathscr{F}^+y$. Furthermore, $x \leq x$ holds for all $x \in \mathscr{P} \cup \mathscr{T}$. Two nodes $x, y \in \mathscr{P} \cup \mathscr{T}$ are *causally related*, if $x \leq y$ or $y \leq x$ holds. We say x is a *causal successor* of y, if $y \leq x$ holds.

Two nodes $x_1, x_2 \in \mathscr{P} \cup \mathscr{T}$ are *in conflict*, denoted $x_1 \# x_2$, if there exist two transitions $t_1, t_2 \in \mathscr{T}$, $t_1 \neq t_2$ with $pre(t_1) \cap pre(t_2) \neq \emptyset$ and $t_i \leq x_i$, $i = 1,2$. A node $x \in \mathscr{P} \cup \mathscr{T}$ is in self-conflict if $x \# x$. Informally speaking, two nodes are in conflict if two transitions exist that share some place in their presets and each node is a causal successor of one of those transitions. Two nodes $x, y \in \mathscr{P} \cup \mathscr{T}$ are *concurrent*, denoted $x||y$, if they are neither causally related nor in conflict.

A Petri net N is *finitely preceded*, if for every node $x \in \mathscr{P} \cup \mathscr{T}$ the set $\{y \in \mathscr{P} \cup \mathscr{T} \mid y \leq x\}$ is finite. That set is the causal history of a node. A Petri net N is *acyclic*, if the directed graph $(\mathscr{P} \cup \mathscr{T}, \mathscr{F})$ is acyclic. The following definitions lead to the definition of a branching process.

An *occurrence net* is a Petri net N with the following properties: N is acyclic, finitely preceded, $\forall p \in \mathscr{P} : |pre(p)| \leq 1$, no transition $t \in \mathscr{T}$ is in self-conflict, and $In = \{p \in \mathscr{P} \mid pre(p) = \emptyset\}$.

A homomorphism from one Petri net to another maps each node to a node such that the preset and postset relations are preserved including the initial marking. Formally, let N_1 and N_2 be two Petri nets. Then a *homomorphism* from N_1 to N_2 is a mapping $h : \mathscr{P}_1 \cup \mathscr{T}_1 \to \mathscr{P}_2 \cup \mathscr{T}_2$ with following properties: $h(\mathscr{P}_1) \subseteq \mathscr{P}_2$ and $h(\mathscr{T}_1) \subseteq \mathscr{T}_2$, for all transitions $t \in \mathscr{T}_1$, h restricted to $pre_1(t)$ is a bijection between $pre_1(t)$ and $pre_2(h(t))$, for all transitions $t \in \mathscr{T}_1$, h restricted to $post_1(t)$ is a bijection between $post_1(t)$ and $post_2(h(t))$, and the restriction of h to In_1 is a bijection between In_1 and In_2. An *isomorphism* is a bijective homomorphism.

A run is represented by a (possibly infinite) firing sequence of transitions. A branching process of a Petri net represents (possibly) multiple runs of the underlying Petri net.

Branching process. A *branching process* of a net N_0 is a pair $B = (N, \pi)$, where N is an occurrence net and π a homomorphism from N to N_0 such that: $(*)$ For all $t_1, t_2 \in \mathscr{T}$: if $pre(t_1) = pre(t_2)$ and $\pi(t_1) = \pi(t_2)$, then $t_1 = t_2$.

An example of a Petri net and a branching process is shown in Fig. 1.

The notion of the set of all reachable markings of a branching process $B = (N, \pi)$ is extended to $\mathscr{R}(B) = \mathscr{R}(N)$. By convention, a branching Process B has the components (N_B, π_B). Throughout this paper, B_1 and B_2 are branching processes of an underlying net N_0. The property $(*)$ of the definition of a branching process ensures that every run of the Petri net is represented at most once. Informally speaking, a run only consists of concurrent and causally related nodes and a node can be part of multiple runs. Nodes that are in conflict, cannot belong to the same run.

Homomorphism on branching processes. A *homomorphism* from B_1 to B_2 is a homomorphism h from N_1 to N_2 such that $\pi_2 \circ h = \pi_1$. The branching processes B_1 and B_2 are *isomorphic* if there exists an isomorphism from B_1 to B_2 which is denoted as $B_1 \cong B_2$.

A natural partial order on branching processes is defined in the following.

Subprocess relation of branching processes. B_1 approximates B_2, denoted by $B_1 \leq B_2$, if there exists an injective homomorphism from B_1 to B_2.

Now we define the unfolding of a net as the maximal branching process that contains all (possibly infinite) runs of a net. Loosely speaking, the unfolding of a net is an acyclic net with the same behaviour as the original net, but where each place and transition has a unique causal history.

Unfolding. The *unfolding* $unf(N_0)$ of a Petri net N_0 is the maximal branching process with respect to the subprocess relation $\leq$ of branching processes. This definition is unique up to isomorphism. We refer to the components of the unfolding as $\mathscr{T}_{unf(N_0)}$, $\mathscr{P}_{unf(N_0)}$, $pre_{unf(N_0)}$, $post_{unf(N_0)}$, and $In_{unf(N_0)}$.

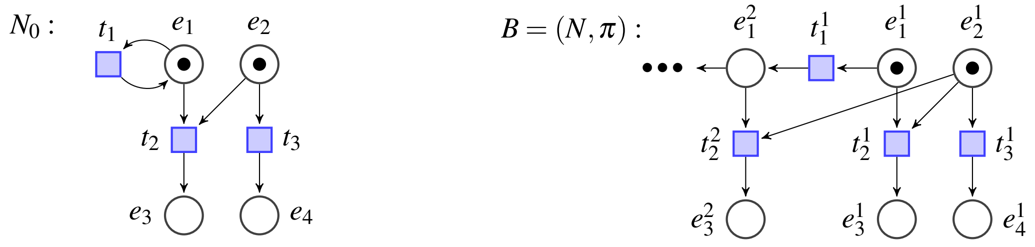


Figure 1: A Petri net N_0 on the left and a branching process $B = (N, \pi)$ of N_0 on the right. Places are shown as circles, transitions as boxes and the preset and postset relations as arrows. The initial marking $\{e_1, e_2\}$ is represented by the black dots, the *tokens*. The homomorphism π from N to N_0 is given as $\pi(e_j^i) = e_j$ and $\pi(t_j^i) = t_j$. The transitions t_1^1 and t_2^1 are in conflict, i.e., $t_1^1 \# t_2^1$, and also $t_2^2 \# t_2^1$, $t_3^1 \# t_2^1$, $t_2^2 \# t_3^1$.

In Fig. 1, the branching process B is the unfolding of the Petri net N_0 assuming that the dots to the left of the place e_1^2 indicate that the branching process continues infinitely in the same way.

We continue with the definition of a Petri game. A Petri game is played on a finite and safe Petri net. Tokens may transit from a system place to an environment place and vice versa.

Petri game. A *Petri game* on an underlying finite and safe Petri net N_0 is a tuple $G = (\mathscr{P}_0^S, \mathscr{P}_0^E, \mathscr{T}_0, pre_0, post_0, In_0, \mathscr{B})$, where the set $\mathscr{P}_0$ of places of N_0 is partitioned into disjoint sets of *system places* $\mathscr{P}_0^S$, and *environment places* $\mathscr{P}_0^E$, and where $\mathscr{B} \in 2^{\mathscr{P}_0}$ is the set of *bad markings*.

A winning strategy of a Petri game is a branching process of the underlying Petri net of the game. A strategy must satisfy 4 properties that are reasonable properties of implementations of processes. Beside a *safety* property, each process must act deterministically, *determinism*, and at least one process must enable a transition if possible, *deadlock avoiding*. Loosely speaking, the *justified refusal* property forces the system players to commit to transitions that are always allowed on their current place. Strategies for Petri games are obtained by cutting out parts of the unfolding.

Winning strategy A *winning strategy* σ of a Petri-game $G = (\mathscr{P}_0^S, \mathscr{P}_0^E, \mathscr{T}_0, pre_0, post_0, In_0, \mathscr{B})$ with underlying Petri-net N_0 is a branching process $\sigma = (N, \pi)$ of N_0 satisfying the following properties:

1. **Justified refusal**: Let $C \subseteq \mathscr{P}$ be a set of pairwise concurrent places and $t \in \mathscr{T}_0$ a transition with $\pi(C) = pre_0(t)$. If no $t' \in \mathscr{T}$ with $\pi(t') = t$ and $pre(t') = C$ exists, then there exists a place $p \in C$ with $\pi(p) \in \mathscr{P}_0^S$, such that $t \notin \pi(post(p))$.
2. **Safety**: For all reachable markings $M \in \mathscr{R}(N)$ it holds that $\pi(M) \notin \mathscr{B}$.
3. **Determinism**: For all $p \in \mathscr{P}$ with $\pi(p) \in \mathscr{P}_0^S$ and for all reachable markings M in N with $p \in M$ there exists at most one transition $t \in post(p)$, which is enabled in M.
4. **Deadlock avoiding**: For all reachable markings M in N there exists an enabled transition, if a transition is enabled in $\pi(M)$ in the underlying Petri-net N_0.

We refer to a token on a system place as a system player, and a token on an environment place as an environment player. The *justified refusal* property ensures that a system player allows all instances of an outgoing transition or no instance at all. The global *safety* property ensures that no bad markings are reachable. The *determinism* property ensures that for each system place at most one transition is enabled in every reachable marking. The *deadlock avoiding* property ensures that the system allows at least one transition in every reachable marking if an enabled transition exists in that marking, e.g. the system players cannot add deadlocks to the existing deadlocks in the Petri net by forbidding transitions.

The *unfolding* $unf(G)$ of a Petri game G is like the unfolding $unf(N_0) = (N, \pi)$ of the underlying Petri net N_0 of G, additionally keeping the distinction between system and environment: a place p in N is a system place if $\pi(p) \in \mathscr{P}_0^S$ and an environment place if $\pi(p) \in \mathscr{P}_0^E$. A winning strategy σ_G of G can be seen as a subprocess of $unf(G)$.

Fig. 2 shows a Petri game with 4 players modelling the control of a room with two doors that must not be opened at the same time so that the two places of the bad marking $\{O^1, O^2\}$ are not reached simultaneously. After receiving a request to open door via transition r^1 the first system player on place S^{12} can decide to proceed with communicating (transition t) or without communicating (transition n^1) to the second system player, then on place S^{22}. On place S^{13} the system player chooses between opening the door (o^1) or denying the request (d^1) for the environment player waiting on W^1. From place S^{14} the system player can close the door it has opened before (c^1). The second system player has the same options after receiving a request via r^2. After that, if both players open their doors the bad marking $\{O^1, O^2\}$ is reached. At least one system player has to deny the request to win the game.

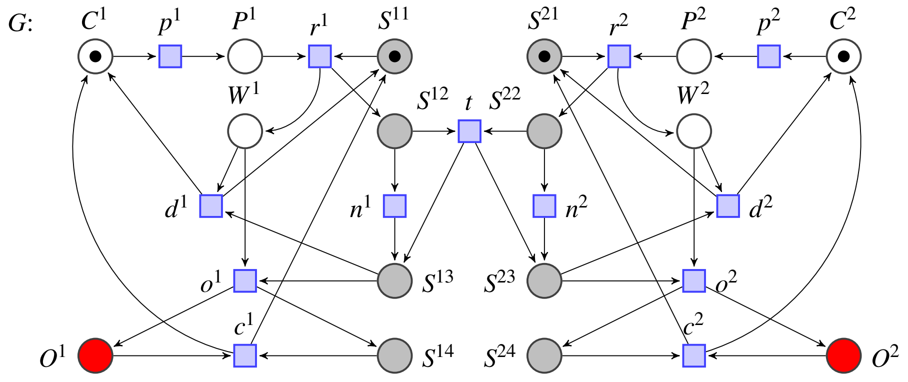


Figure 2: A Petri game G: the grey places belong to the system players and the white places to the environment players. There are two doors and two system players taking requests to open the door. After a request the system players decide whether to communicate and open their door afterwards. The bad marking $\{O^1, O^2\}$ is reached if the system players decide to open both doors simultaneously.

Fig. 3 shows an initial part of the unfolding of the Petri game in Fig. 2. The parts that are not greyed out are the initial parts of a winning strategy. Here, the system player agree on communicating via t_1. The first player decides to open its door via transition o_2^1 while the other door remains closed via transition d_2^2. After the transition c_2^1 the first door is closed again. The following is not shown in Fig. 3 anymore: after both players have received another opening request via r_2^1 and r_2^2, respectively, the winning strategy could continue opening door 2 and keeping door 1 closed since the different causal histories allow different decisions.

According to the definition of a winning strategy it is also possible that a winning strategy denies all requests to open a door. The example is chosen with foresight for the content in Section 3.

3 Decidability Results

In this section, we show that the synthesis problem for Petri games with up to 4 players under a global safety condition is decidable in non-deterministic exponential time (NEXP), and NP-complete in the 2-player case. In [11], a related result is shown that the synthesis problem for 4-process control games

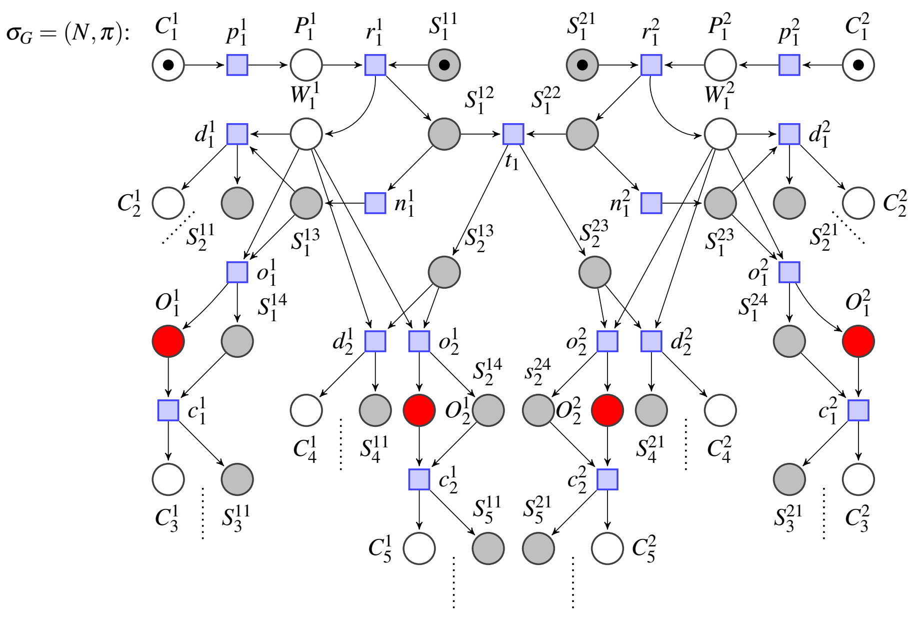


Figure 3: An initial part of the unfolding of the Petri game G in Fig. 2. The nodes that are not greyed out form an initial part of a winning strategy σ_G where door 1 gets opened and door 2 remains closed after the system players have communicated. The homomorphism π is defined analogously to that in Fig. 1.

with a local termination condition is decidable. The translation [1] of this result to Petri games gives a decidability result for Petri games with 4 players with a local termination condition, without process generation and deletion. There are no complexity bounds for the 4-player case given in [11], and the translation to Petri games already generates an exponential blow up in the number of nodes [1].

A Petri game G is called a *K-player Petri game*, $K \in \mathbb{N}$, if and only if for all reachable markings $M \in \mathscr{R}(G)$ it holds that $|M| \leq K$. Two-player Petri games can be seen as a natural generalisation of infinite games on graphs with a safety winning condition. NP-hardness is established by a reduction of the 3-SAT problem to 2-player Petri games.

Lemma 3.1. *There exists a polynomial-time reduction of the 3-SAT problem to the synthesis problem of two-player Petri games.*

Proof. Let $F = (x_1^1 \vee x_2^1 \vee x_3^1) \wedge \ldots \wedge (x_1^n \vee x_2^n \vee x_3^n)$ be an instance of the 3-SAT problem in conjunctive normal form where all clauses consist of exactly three literals and where x_j^i, $i = 1, \ldots, n$ and $j = 1,2,3$, is a positive or negative (with overline) literal from the set $\{x_1, \bar{x}_1, \ldots x_m, \bar{x}_m\}$. Fig. 4 shows the Petri game of F. If F is satisfiable, the top system player chooses the transitions according to the boolean assignment that satisfies F, for example $\bar{x}_1$ if the truth value of x_1 is false under the boolean assignment. The bottom system player chooses the literal that is true under the boolean assignment in each clause, for example if $x_1^1 = \bar{x}_1$, she may choose x_1^1 since the top system player chooses it. Both system players do not know how when the transition of the other player are fired such that it has to be correct for all possible orders of execution, for example the top player could have chosen the truth value for x_n already before the bottom player chose the literal for the first clause.

Conversely, the top player's choices yield a boolean assignment that satisfies F if the Petri game shown has a winning strategy since the bottom player chooses one literal of each clause that must be true

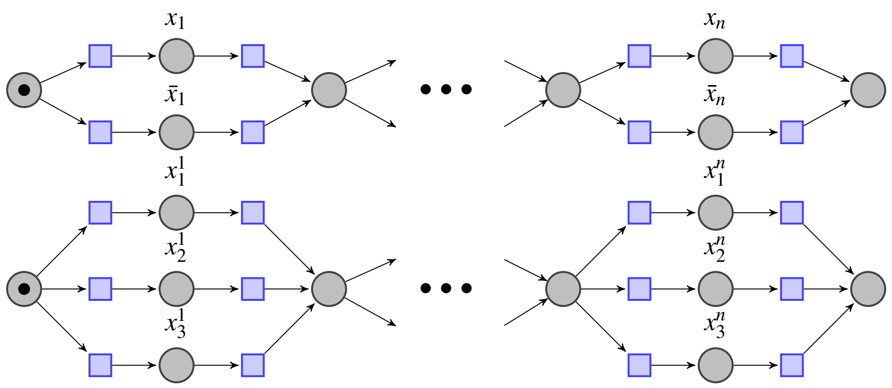


Figure 4: 3-SAT problem F as a Petri game. The top system player chooses sequentially the truth value for $x_1,\ldots x_n$. The bottom system player has to choose a literal for each clause according to the truth values chosen by the top player. The bad markings are $\{\{x_k,x_j^i\} \mid x_j^i = \bar{x}_k\} \cup \{\{\bar{x}_k,x_j^i\} \mid x_j^i = x_k\}$. So every time the bottom player chooses a literal that is not chosen by the top system player a bad marking is reached, e.g. the top player chooses x_1 and the bottom player x_3^1 where $x_3^1 = \bar{x}_1$.

under the boolean assignment. □

Note that the 2-player case is NP-hard even without an environment player. The matching upper bound is established later, along with the complexity of the 4-player case.

The idea of solving 4-player Petri games is to find game states where a winning strategy can be repeated and still win. A few definitions are necessary to formalise repeating a part of a winning strategy. The following definitions about branching processes are equivalent to those in [5].

Cut. The reachable markings in a branching process B are called *cuts*. A cut is a maximal set of pairwise concurrent places [7].

Future of a cut. We define the branching process $Fut(B,C)$ of a cut $C \subseteq \mathscr{P}_B$ as follows: $\mathscr{P}_{Fut(B,C)} \cup \mathscr{T}_{Fut(B,C)} = \{x \in \mathscr{P}_B \cup \mathscr{T}_B \mid \forall p \in C : p \leq x \vee p||x\}$, the mappings *preset* and *post* are the mappings pre_B and $post_B$ restricted to $\mathscr{T}_{Fut(B,C)}$, and $In_{Fut(B,C)} = C$. Generally, the *future* $Fut(B,C)$ is not a branching process of the underlying net N_0 of B (i.e. $\pi_B : \mathscr{T}_B \cup \mathscr{P}_B \to \mathscr{T}_0 \cup \mathscr{P}_0$) but it is a branching process of the net $(\mathscr{P}_0, \mathscr{T}_0, pre_0, post_0, \pi_B(C))$, i.e. $Fut(B,C)$ is a branching process of N_0 if $\pi_B(C) = In_0$. (This definition is equal to the definition of $\Uparrow$*Configuration* in [5].)

Informally speaking, the definition of the future of a cut coincides with the intuition that it is the branching process that follows after that cut.

Last known cut and last known marking. The *last known cut* $lkc(t)$ of a transition t of a branching process B is defined as $lkc(t) = \{p \in \mathscr{P}_B \mid p \not< t \wedge \forall t' \in pre_B(p) : t' \leq t\}$. Informally speaking, this cut is reached by firing all transitions that are causal predecessors of t. The *last known marking* $lkm(t)$ is defined as $\pi_B(lkc(t))$, which is the marking reached in the underlying Petri net. (The *lkc* is the cut of a *local configuration* [5])

A *cut* and *glue* operation formalises the process of copying the future of one cut to another cut of a strategy. Later, the actual requirements for when to copy are defined. In the following, B and B' are branching processes of (possibly different) nets.

Cut. $B - B' = (\mathscr{P}_B \setminus (\mathscr{P}_{B'} \setminus In_{B'}), \mathscr{T}_B \setminus \mathscr{T}_{B'}, pre_B \upharpoonright_{\mathscr{T}_{B-B'}}, post_B \upharpoonright_{\mathscr{T}_{B-B'}}, In_B)$

If B' is the future of a cut of B, the cut operation removes B' from B leaving only the initial marking of B' in B.

Glue. $B+B' = (\mathscr{P}_B \cup \mathscr{P}_{B'}, \mathscr{T}_B \cup \mathscr{T}_{B'}, pre_{B+B'}, post_{B+B'}, In_B)$, where $pre_{B+B'}(t) = \begin{cases} pre_B(t) & t \in \mathscr{T}_B \\ pre_{B'}(t) & t \in \mathscr{T}_{B'} \end{cases}$, and analogously $post_{B+B'}$.

If the initial marking of B' is a cut in B, B' gets glued to that cut. Generally, $B-B'$ and $B+B'$ are not branching processes. However, if there are two cuts $C_1, C_2 \subseteq \mathscr{P}_B$ with $\pi_B(C_1) = \pi_B(C_2)$ cutting out the future of C_2 and glueing an isomorphic copy (this is just a necessary renaming) of the future of C_1 to C_2 yields a branching process.

Cut and glued branching process. Let $Fut(B,C_1)'$ be an isomorphic copy of $Fut(B,C_1)$, $Fut(B,C_1)' \cong Fut(B,C_1)$, such that $In_{Fut(B,C_1)'} = C_2$ and $(\mathscr{P}_B \cup \mathscr{T}_B) \cap (\mathscr{P}_{Fut(B,C_1)'} \cup \mathscr{T}_{Fut(B,C_1)'}) = C_2$. The cut and glued branching process $B_{C_1 \to C_2}$ is the branching process $B_{C_1 \to C_2} = B - Fut(B,C_2) + Fut(B,C_1)'$. This definition is unique up to isomorphism.

Definition 3.1 (Imitable cuts)**.** Let σ be a winning strategy of a Petri game G. A cut C_2 is imitable by a cut C_1 if $\sigma_{C_1 \to C_2}$ is a winning strategy.

Now we define the actual cuts that are imitable. The first kind of these cuts are cuts where a subset of players take transitions without communicating to the remaining players until all players of this subset synchronise at a joint transition for the second time. In the following, we fix σ as a winning strategy of a Petri game G.

Definition 3.2 (Partial repetition cuts)**.** Let $t_1, t_2 \in \mathscr{T}_\sigma$. The cuts $lkc(t_1)$ and $lkc(t_2)$ are *partial repetition cuts*, denoted $prc(t_1,t_2)$, if $lkm(t_1) = lkm(t_2) \land lkc(t_1) \setminus post(t_1) = lkc(t_2) \setminus post(t_2)$. The partial repetitions cuts $prc(t_1,t_2)$ are called a *loop*, denoted $prc_{loop}(t_1,t_2)$ if $t_1 < t_2$.

Lemma 3.2 (Partial repetition cuts are imitable)**.** *For transitions $t_1, t_2 \in \mathscr{T}_\sigma$, $prc(t_1,t_2)$ implies that $lkc(t_2)$ is imitable by $lkc(t_1)$ and vice versa.*

Proof by contradiction. We show that $\sigma_{lkc(t_1) \to lkc(t_2)} = \sigma - Fut(\sigma, lkc(t_2)) + Fut(B, lkc(t_1))'$ is a winning strategy. Suppose that there exists a cut $C \subseteq \mathscr{P}_{\sigma_{lkc(t_1) \to lkc(t_2)}}$ such that one of the properties of the winning strategy is violated.

We distinguish two cases: The first case is that there exists a place $p \in C$ such that $t_2 \leq p$. Then, $C \subseteq \mathscr{P}_{Fut(\sigma_{lkc(t_1) \to lkc(t_2)}, lkc(t_2))}$, holds. Since $Fut(\sigma_{lkc(t_1) \to lkc(t_2)}, lkc(t_2)) = Fut(\sigma, lkc(t_1))' \cong Fut(\sigma, lkc(t_1))$ it follows directly that σ is not a winning strategy if $\sigma_{lkc(t_1) \to lkc(t_2)}$ is not winning.

The second case is that there exists no place $p \in C$ such that $t_2 \leq p$, i.e. $\forall p \in C : p \leq t_2 \lor p \# t_2 \lor p || t_2$ holds. Since the last known cuts of t_1 and t_2 are equal except for $post(t_1)$ and $post(t_2)$ the branching processes $Fut(\sigma, lkc(t_2))$ and $Fut(\sigma, lkc(t_1))$ are isomorphic up to the nodes that are causal successors of t_1 and t_2 respectively. This means that all transitions and places that are not causal successors of t_2 are only renamed by constructing $\sigma_{lkc(t_1) \to lkc(t_2)}$. Since there is no place in C that is a causal successor of t_2 there is no transition enabled in C that could have been added or removed. It follows that σ is not a winning strategy, if $\sigma_{lkc(t_1) \to lkc(t_2)}$ is not a winning strategy. □

In Fig. 5 is an example of partial repetition cuts. Note that not all of those partial repetition cuts are loops as the players can reach the same places in the underlying Petri net by taking different transitions. Partial repetition cuts are defined regardless of the number of players in a Petri game, i.e, these cuts are imitable in any Petri game.

Maximally repeated strategy. Since the nodes of a branching process are countable we can construct by induction over the partial repetition cuts of a winning strategy σ a winning strategy σ^{pr} where $prc(t_1,t_2)$ implies that $Fut(\sigma^{pr}, t_1) \cong Fut(\sigma^{pr}, t_2)$. σ^{pr} denotes a *maximally repeated strategy* of σ.

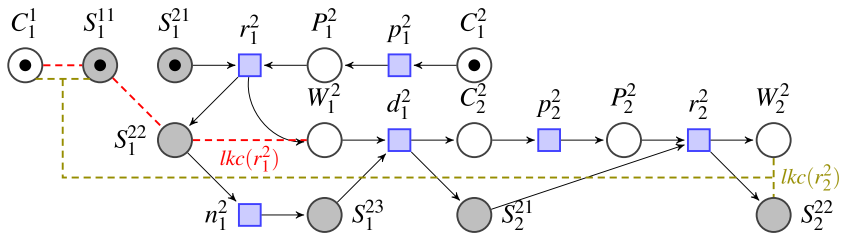


Figure 5: Example of partial repetition cuts. Assume that this is a part of a winning strategy where the second system player denies (d_1^2) the first request (r_1^2) to open the door without communicating with the other system player via transition t. Here, the last known cuts of r_1^2 and r_2^2 are $lkc(r_1^2) = \{C_1^1, S_1^{11}, S_1^{22}, W_1^2\}$ and $lkc(r_2^2) = \{C_1^1, S_1^{11}, S_2^{22}, W_2^2\}$, so $lkm(r_1^2) = lkm(r_2^2)$. Thus, a partial repetition cut (that is a loop) occurs $prc_{loop}(r_1^2, r_2^2)$.

The partial repetition cuts have a useful implication for the case with up to 4 players: if two players do not take any joint transition with one of the two remaining players they will get to a partial repetition cut eventually. The idea of the following definition of a synchronisation segment is based on this implication. Informally speaking, a synchronisation segment describes the part starting from the last known cut of a transition t until all other players are causal successors of this transition or the strategy can be repeated due to the partial repetition cuts. Here, the idea of when to repeat a strategy is that if all players play identically starting from the last known cut of transition t until they get to know of transition t the strategy can be repeated after transition t.

Definition 3.3 (Synchronisation segment)**.** The *synchronisation segment $Seg(t)$* of a transition $t \in \mathscr{T}_B$ of a branching process B is defined as a smallest branching process (with respect to $\leq$) such that
(1.) $Seg(t) \leq Fut(B, lkc(t))$ and
(2.) $\forall t' \in \mathscr{T}_{Fut(B,lkc(t))} : t' \notin \mathscr{T}_{Seg(t)} \Rightarrow$ (a) $(\forall p \in pre_B(t') : t \leq p) \vee$
(b) $(\exists t_1, t_2, t_3 \in \mathscr{T}_{Seg(t)} : prc_{loop}(t_1, t_2) \wedge t_3 \leq t_2 \wedge prc(t', t_3))$

For a transition $t' \in Fut(B, lkc(t))$ that is not in $\mathscr{T}_{Seg(t)}$, all places in its preset are causal successors of t (2.a) or there is a transition t_3 with $prc(t', t_3)$ within a loop (2.b). So, the parts of a winning strategy that get repeated due to partial repetition cuts are included in the synchronisation segment. An example of a synchronisation segment is shown in Fig. 6 and Fig. 7. *Synchronisation equivalent cuts* are those cuts where the synchronisation segments are isomorphic.

Definition 3.4 (Synchronisation equivalent cuts)**.** Let σ^{pr} be a maximally repeated winning strategy of a Petri game G and $t_1, t_2 \in \mathscr{T}_{\sigma^{pr}}$. The cuts $lkc(t_1)$ and $lkc(t_2)$ are *synchronisation equivalent cuts*, denoted $sqc(t_1, t_2)$, if $Seg(t_1) \cong Seg(t_2)$. $sqc(t_1, t_2)$ is called an *s-loop*, denoted sqc_{loop}, if $t_1 < t_2$.

Now, we show that a winning strategy can be repeated after synchronisation equivalent cuts.

Lemma 3.3 (Synchronisation equivalent cuts are imitable)**.** *For transitions $t_1, t_2 \in \mathscr{T}_{\sigma^{pr}}$, $sqc(t_1, t_2)$ implies that $lkc(t_2)$ is imitable by $lkc(t_1)$ and vice versa.*

Proof. We show that $\sigma^{pr}_{lkc(t_1) \rightarrow lkc(t_2)} = \sigma^{pr} - Fut(\sigma, lkc(t_2)) + Fut(\sigma, lkc(t_1))'$ is a winning strategy. Since $Seg(t_1) \cong Seg(t_2)$ the construction is only a renaming for the nodes in the synchronisation segment and for those that get repeated due to the partial repetition cuts. Thus, only nodes that are causal successors of t_2 may be removed or added. Now, the proof works in the same way as the proof of Lemma 3.2, that partial repetition cuts are imitable. □

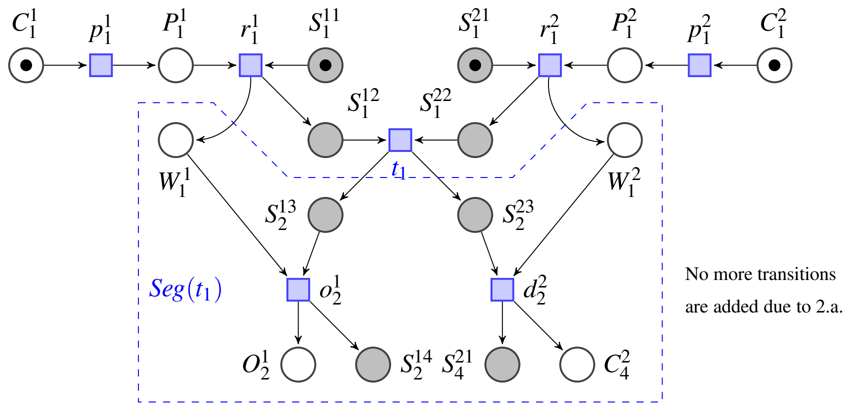


Figure 6: A subprocess of σ_G of Fig. 3 is shown. The nodes inside the dashed, blue box are the nodes of the synchronisation segment $Seg(t_1)$. The initial marking is $In_{Seg(t_1)} = \{W_1^1, S_2^{13}, S_2^{23}, W_1^2\}$. This segment ends after the transitions o_2^1 and d_2^2 since $t_1 < O_2^1, S_2^{14}, S_4^{21}, C_4^2$. So, if σ_G repeats to open the first door and keeping the second door closed after taking the transition t the winning strategy could repeat itself due to synchronisation equivalent cuts.

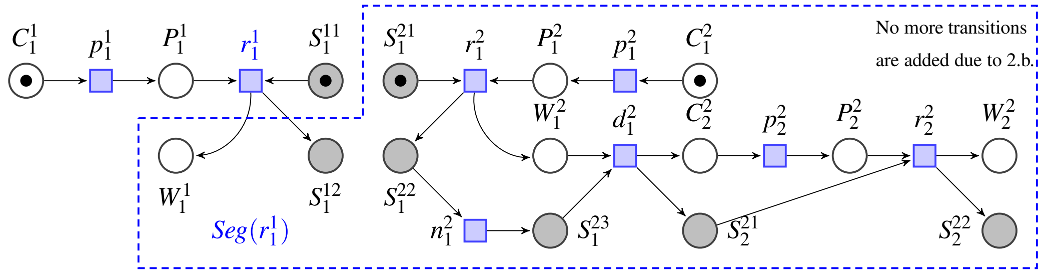


Figure 7: A branching process of the Petri game G of Fig. 2 is shown. The nodes inside the dashed, blue box are the nodes of the synchronisation segment $Seg(r_1^1)$.

We show that 4-player Petri games can be solved in NEXP with the help of Lemma 3.2 and Lemma 3.3. A winning prefix of sufficient size is guessed from which a winning strategy can be constructed. A branching process B is a *winning prefix* if there exists a winning strategy σ with $B \leq \sigma$.

Theorem 3.4. *The synthesis problem of 2-player Petri games is NP-complete and in NEXP for 3- and 4-player Petri games.*

Proof. Structure of this proof: from the 2-player case over the 3-player case to the 4-player case the size of a winning prefix is determined to guarantee that a winning strategy can be constructed by repeating the futures of imitable cuts. A prefix of appropriate size is guessed and it is checked if it is winning. The prefix does not contain any causal successors of transitions t_2 if $\exists t_1 : prc_{loop}(t_1,t_2) \vee sqc_{loop}(t_1,t_2)$. Also, it does not contain causal successors of transitions t_4, if $\exists t_1,t_2,t_3 : t_3 \leq t_2 \wedge ((prc(t_3,t_4) \wedge prc_{loop}(t_1,t_2)) \vee (sqc(t_2,t_3) \wedge sqc_{loop}(t_1,t_2)))$. Let T be the number of transitions of the Petri game.

If a player does not take any joint transition she eventually repeats a place she lays on, i.e. she reaches a partial repetition cut. This occurs after at most T transitions. Otherwise, the two players take a joint transition after at most T transitions. Since there are at most T different joint transitions, a winning prefix from which a winning strategy can be constructed is at most of size $\mathcal{O}(T^2)$. To check if the guessed prefix is winning, we have to check all reachable markings for the winning properties. In a safe Petri net with T^2 transitions and at most two tokens there are at most T^4 reachable markings. Thus, guessing a winning prefix and checking if it is winning takes time in $\mathcal{O}(|\mathcal{T}|^4)$, which shows together with Lemma 3.1 that the synthesis problem for two-player Petri games is NP-complete.

In the 3-player case, it holds for the same reasons as in the 2-player case that each pair of two players reaches partial repetition cuts after at most T^2 transitions without taking a joint transition with the remaining third player. The remaining third player can take up to T transitions herself before reaching a partial repetition cut or taking a joint transition. Therefore, a synchronisation segment has at most $T^2 + T$ transitions resulting in at most $2^{(T^2+T)^2}$ non-isomorphic synchronisation segments. Therefore, a synchronisation equivalent cut is always reached after at most $2^{(T^2+T)^2}$ transitions such that the winning prefix can be extended step by step by reaching synchronisation equivalent cuts again and again. Thus, the size of a winning prefix is in $\mathcal{O}(2^{T^4})$. To check if the prefix is winning takes polynomial time in its size and it follows that the 3-player case is decidable in NEXP.

In a 4-player Petri game, it also holds that each pair of two players reaches partial repetition cuts after at most T^2 transitions without taking a joint transition with one of the two remaining players. So, there are two pairs of players that can take T^2 transitions until they reach partial repetition cuts or one player of each pair communicate with each other. This leaves the other two players until they reach partial repetition cuts or take a joint transition again after T^2 transitions. Thus, there are at most 2^{3T^2} non-isomorphic synchronisation segments. Therefore, the size of a winning prefix of a 4-player Petri game that needs to be checked if it is winning is in $\mathcal{O}(2^{6T^2})$. So the bound is in NEXP. Note that if more than 2 players take a joint transition the size of the synchronisation segments decreases. $\square$

This construction does not work for the 5-player case. The problem that occurs is tricky to see. Assume there is a group of 3 players and a group of 2 players that take joint transitions repeatedly within their group, only two players participating at a time. The group of 2 players get to partial repetition cuts or one of them takes a joint transition with one player of the other group, while the group of 3 players might not get to partial repetition cuts. After two players, one from each group, have taken a joint transition the problem occurs: the remaining 3 players that did not participate in that joint transition might form a new group of 3 players that do not reach partial repetition cuts. This cannot occur in the 4-player case since there is only a pair of players left.

4 Undecidability Result

In this section, a tiling problem is reduced to the synthesis problem of a 6-player Petri game with a global safety condition. The tiling problem used here is the ω bipartite colouring problem (ω-BCP). The undecidable ω Post correspondence problem (ω-PCP) is reducible to the ω-BCP. Later, the undecidability result can be simplified to local safety as a winning condition. The reduction is similar to the work in [12], where the (normal) Post correspondence problem is reduced to a colouring problem followed by a reduction to the synthesis problem of 6-process control games with local termination as a winning condition.

Employing the translation of control games into Petri games in [1], the 6-process control games yield 6-player Petri games (with exponentially many additional nodes) for local termination as a winning condition. In the obtained Petri game, all players alternate in their roles as environment and system players. Instead, we present a direct construction of 6-player Petri games with at most 3 simultaneous system players. Later, we show that the number of system players can be even further reduced to 2.

Definition 4.1 (ω bipartite colourings)**.** Let C be a finite set of colours. An ω *bipartite colouring* is a mapping $f : \mathbb{N}^2 \mapsto C$. The initial colour is $f(1,1)$. We define three subsets of C^2 called the patterns induced by f:

- the *diagonal patterns* of f are all pairs $\{(f(x,y), f(x+1,y+1)) \mid (x,y) \in \mathbb{N}^2)\}$
- the *horizontal patterns* of f are all pairs $\{(f(x,y), f(x+1,y)) \mid (x,y) \in \mathbb{N}^2)\}$
- the *vertical patterns* of f are all pairs $\{(f(x,y), f(x,y+1)) \mid (x,y) \in \mathbb{N}^2)\}$

We define a *colouring constraint* as a 4-tuple (C_i, DP, HP, VP), where C_i is the set of *initial* colours, DP a set of diagonal patterns, VP a set of vertical patterns and HP a set of horizontal patterns. DP, VP and HP are called *forbidden patterns*. A colouring f *satisfies* a colouring constraint if its initial colour is in C_i and no diagonal pattern of f is in DP, no vertical pattern of f is in VP and no horizontal pattern of f is in HP.

ω-BCP. Given a finite set of colours C and colouring constraints (C_i, DP, VP, HP), decide whether there exists an ω bipartite colouring that satisfies the colouring constraints. This problem is a variation of the standard tiling problem and it is also undecidable. In the following we define a Petri game for which a winning strategy exists if and only if a given ω-BCP has a solution. In this Petri game shown in Fig. 8, there are two identical parts, a top part and a bottom part, each consisting of 3 players. The idea is that the number of *rounds* played in the lower and upper parts refer to the x and y coordinates of a tile, respectively, so that the system players choose a colour for each tile. Each colour choice requires one player from each part of the Petri game, so there can be a maximum of 3 colour choices in one run of the Petri game. If the colours chosen refer to a forbidden pattern a bad marking is reached. As the bad markings have to be defined on the Petri net and not on the unfolding, we define when two system players are in the same round or when a player is one round ahead or one round behind to be able to check for forbidden patterns.

Compared rounds of two players. In the Petri net N_{CP} of Fig. 8, we compare the rounds of two players a and b, both from the same part. We say $a, b \in \{0,1,2\}$ are in the same round if and only if for their places e^X_{aj} and e^X_{bk}, $X \in \{T, B\}$, $j,k \in \{0,1,2\}$ or $j,k \in \{3,4\}$ or $j,k \in \{5,2\}$ holds. We say that a is one round ahead of b (or b is one round behind of a) if and only if ($j \in \{3\}$ and $k \in \{2\}$) or ($j \in \{5\}$ and $k \in \{4\}$) . Other combinations of indices are not possible.

Definition 4.2 (Petri game of an ω Bipartite colouring Problem)**.** Let CP be an ω-BCP with colours C and constraints C_i, DP, VP and HP. The components of the Petri net of the Petri game G_{CP} are the components of N_{CP} from Fig. 8. The bad markings of the Petri game G_{CP} are defined as follows:

$\mathscr{B} = \mathscr{B}_{Same} \cup \mathscr{B}_{DP} \cup \mathscr{B}_{HP} \cup \mathscr{B}_{VP} \cup \mathscr{B}_{init}$, where
$\mathscr{B}_{Same} = \{\{e_{a^T j - a^B j'}, e_{b^T k - b^B k'}, (c_u, a), (c_v, b)\} \mid$
a^T and b^T are in the same round and a^B and b^B are in same the round $, c_u \neq c_v\}$,
$\mathscr{B}_{DP} = \{\{e_{a^T j - a^B j'}, e_{b^T k - b^B k'}, (c_u, a), (c_v, b) \mid$
a^T and a^B are one round ahead of b^T and b^B respectively $, (c_v, c_u) \in DP\}$,
$\mathscr{B}_{VP} = \{\{\{e_{a^T j - a^B j'}, e_{b^T k - b^B k'}, (c_u, a), (c_v, b)\} \mid$
a^T is a round ahead of b^T and a^B and b^B are in the same round $, (c_v, c_u) \in VP\}\}$,
$\mathscr{B}_{HP} = \{\{\{e_{a^T j - a^B j'}, e_{b^T k - b^B k'}, (c_u, a), (c_v, b)\} \mid$
a^B is a round ahead of b^B and a^T and b^B are in the same round $, (c_v, c_u) \in HP\}\}$,
$\mathscr{B}_{init} = \{\{e_{a^T j - a^B j'}, (c_u, a)\} \mid j, j' \in \{0,1\}, c_u \notin C_i\}$.

In addition to the colouring constraints, the bad markings contain the set $\mathscr{B}_{Same}$ to ensure that system players must choose the same colour for two checks whenever the checks occur in the same round for

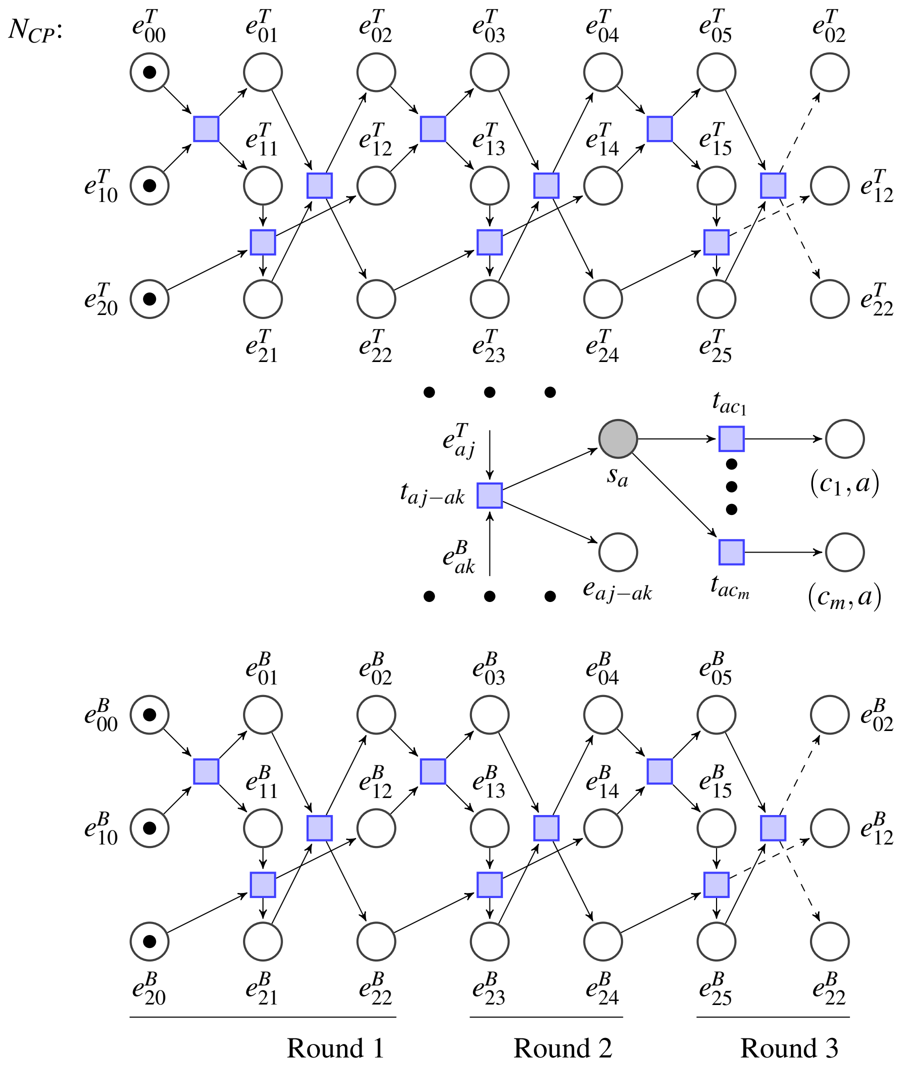


Figure 8: Petri net N_{CP} of the Petri game G_{CP} of an ω-BCP. We have two structurally identical parts consisting of 3 environment players, a top part and a bottom part. Two of the players synchronise at each transition in these parts. After firing 3 such transitions, each player has synchronised with the other two players in its part. We define such a block of 3 transitions as a *round*. Each part has 3 rounds. After the third round the players return to the places they were in before the second round. This means that we have an initial round and then the players alternate between the second and third rounds. As the game progresses, the nodes have exact information in the unfolding about how many rounds they have played. In the middle part, the environment players can take a *check* transition t_{aj-ak}, where $a \in \{0,1,2\}$ and $j,k \in \{0,\dots,5\}$ and $pre(t_{aj-ak}) = \{e^T_{aj}, e^B_{ak}\}$. After that, the system player on place s_a has to choose a colour. Based on the number of rounds x and y played in the lower part and upper part, respectively, the system player has to determine the colour $f(x,y)$ of the given ω-BCP.

both players. To define a colouring from a strategy of the Petri game, we define the number of rounds played in the unfolding.

Number of rounds. Let $unf(G_{CP})$ be the unfolding of G_{CP} of Fig. 8. The *number of rounds* played in part X, $X \in \{T,B\}$ in a place $p \in \mathscr{P}_{unf(G_{CP})}$ is defined as $r^X(p) = max(\lceil |\{t \in \mathscr{T}_{unf_{CP}} \mid t \leq p$ and t is a transition in part $X\}|/3\rceil, 1)$

This means that the number of rounds is incremented after a player has taken two transitions, starting from round 1. The divisor is 3 because the players also get the knowledge of the synchronisation of the other two players in their part. The number of rounds is not increased after a check transition.

Theorem 4.1 (Characterization of ω-BCP as a Petri game)**.** *The ω-BCP is reducible to the synthesis Problem of 6-player Petri games.*

Proof. Let CP be a colouring problem and G_{CP} be the Petri game from Def. 4.2. We show that there exists a solution for CP if and only if there exists a winning strategy for G_{CP}.

We start by assuming that there is a solution $f : \mathbb{N} \times \mathbb{N} \mapsto C$ for the colouring problem CP. After the environment player has chosen a check transition the system player must choose a colour in place s_a, $a \in \{0,1,2\}$. Since the system player knows the entire causal history, she knows the number of rounds played in the top and the bottom part of the Petri game. Let $r^T(s_a) = y$ denote the number of rounds in the top part and $r^B(s_a) = x$ the number of rounds in the bottom part. Then, the winning strategy for the system players is to choose the colour $f(x,y)$. This way no bad marking is reachable. The bad markings in $\mathscr{B}_{Same}$ are avoided because $f(x,y)$ is unique. The bad markings in $\mathscr{B}_{DP}, \mathscr{B}_{VP}$ and $\mathscr{B}_{HP}$ are avoided because f avoids all forbidden patterns. Finally, the bad markings $\mathscr{B}_{init}$ are avoided because $f(1,1)$ is an initial colour.

Now, we assume that there is a winning strategy for the Petri game G_{CP}. We define the colouring $f : \mathbb{N} \times \mathbb{N} \mapsto C$ derived from the winning strategy as follows: $f(x,y) = c$ if there exists a system place $s_a \in \mathscr{P}_{unf(N_{CP})}$ with $r^B(s_a) = x$ and $r^T(s_a) = y$ and $t_{ac} \in post(s_a)$.

We show that for every colour that a system player has to choose, there are sequences of checks such that all colouring constraints are met. In these sequences of checks, any two consecutive checks can be carried out concurrently while the system players do not know whether the other check is the previous check or the next check, so the winning strategy must play correctly for both checks. Informally speaking, this leads to infinite sequences of dependencies as a check depends on the choice of the colour of the previous check and again that check depends on its previous check and so on.

In Fig. 9, it is crucial to see that there are concurrent places between different rounds such that we can always find at least one suitable check for all diagonal, vertical and horizontal patterns. To check the horizontal pattern of $(f(1,1), f(2,1))$ for example, the checks t_{00-03} and t_{20-22} are fired. The places in the upper part are e^T_{00} and e^T_{20} respectively, which are both in the first round. The places in the lower part are e^B_{03} and e^B_{22} respectively, where e^B_{03} is in the second round and e^B_{22} in the first round. These two checks can be performed concurrently since e^T_{00} and e^T_{20} are concurrent, as are e^B_{22} and e^B_{03}. For the initial colour we have the extra initial round which does not get repeated.

The colouring f is well-defined as the set of bad markings $\mathscr{B}_{Same}$ ensures that every two colours chosen when both top rounds and both bottom rounds are the same the colours chosen also must be the same in a winning strategy. This can be seen with the help of Fig. 9, too. There, we can find a sequence of checks throughout one round such that the colour chosen at the beginning of that round (e.g. e^B_{03}) is still the same colour as chosen at the end of that round (e.g e^B_{04}) (assuming that the top round does not change too). □

It is easy to see that the Petri game presented here performs at most 3 checks in every possible

	e^X_{00}	e^X_{10}	e^X_{20}	e^X_{01}	e^X_{11}	e^X_{21}	e^X_{02}	e^X_{12}	e^X_{22}	e^X_{03}	e^X_{13}	e^X_{23}	e^X_{04}	e^X_{14}	e^X_{24}
e^X_{00}		‖	‖												
e^X_{10}	‖		‖												
e^X_{20}	‖	‖		‖	‖										
e^X_{01}			‖		‖	‖		‖							
e^X_{11}			‖	‖											
e^X_{21}				‖				‖							
e^X_{02}								‖	‖						
e^X_{12}				‖		‖	‖		‖						
e^X_{22}							‖	‖		‖	‖				
e^X_{03}									‖		‖	‖		‖	
e^X_{13}									‖	‖					
e^X_{23}										‖				‖	
e^X_{04}														‖	‖
e^X_{14}										‖		‖	‖		
e^X_{24}													‖	‖	

Figure 9: Table of concurrent places of the first two rounds in the unfolding of the Petri game G_{CP}. Empty cells denote that the places are causally related. Filled cells denote that the places are concurrent. The pattern continues identically. With the help of this table we can see the sequences of checks to ensure that no bad markings are reached in the Petri game. For example, to check that the initial colour chosen after the transition t_{00-00} and after t_{22-00} is the same, we can perform the following sequence of checks of the indices of the places in the upper part: $00-20-01-12-22$. The indices of the place in the lower part remain 00 here. To carry out two consecutive checks in the same play they need to be concurrent. We can check this in the table by going in a row from 00 to 20, then in the column from 20 to 01, then again checking in the row and so on. Note that the place of the bottom player can also change in these sequences.

sequence of transitions. Therefore this Petri game has at most 3 system players. For the undecidability result it is sufficient to have 2 concurrent checks. Therefore, the number of system players can be further reduced to 2 by adapting the Petri game. This could be done by adding two environment players that are consumed performing a check. Furthermore, by adding a transition for each bad marking to a bad place, the set of bad markings can be simplified to a set of bad places, the local safety condition.

5 Conclusions

Our contribution to the synthesis problem of distributed systems is that this problem is undecidable for 6-player Petri games under a local safety condition and that two system processes are sufficient to obtain undecidability. This adds neatly to the result in [7], where Petri games with one system player are solved in exponential time. Furthermore, the synthesis problem for 2-player Petri games belongs to the class of NP-complete problems, and we provide a non-deterministic exponential upper bound for this problem for Petri games with a variable number of players up to 4 under a global safety condition. This is a new class of Petri games for which the synthesis problem is decidable. The case of 5 players remains open.

References

[1] Raven Beutner, Bernd Finkbeiner & Jesko Hecking-Harbusch (2019): *Translating Asynchronous Games for Distributed Synthesis*. In Wan J. Fokkink & Rob van Glabbeek, editors: *30th International Conference on*

Concurrency Theory, CONCUR 2019, *LIPIcs* 140, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 26:1–26:16, doi:10.4230/LIPIcs.CONCUR.2019.26.

[2] Roderick Bloem, Sven Schewe & Ayrat Khalimov (2017): *CTL* synthesis via LTL synthesis*. In Dana Fisman & Swen Jacobs, editors: *Proceedings Sixth Workshop on Synthesis, SYNT@CAV 2017*, *EPTCS* 260, pp. 4–22, doi:10.4204/EPTCS.260.4.

[3] Alonzo Church (1957): *Applications of recursive arithmetic to the problem of circuit synthesis*. *Summaries of the Summer Institute of Symbolic Logic* 1, pp. 3–50.

[4] Joost Engelfriet (1991): *Branching Processes of Petri Nets*. *Acta Inf.* 28(6), pp. 575–591, doi:10.1007/BF01463946.

[5] Javier Esparza, Stefan Römer & Walter Vogler (2002): *An Improvement of McMillan's Unfolding Algorithm*. *Formal Methods Syst. Des.* 20(3), pp. 285–310, doi:10.1023/A:1014746130920.

[6] Bernd Finkbeiner (2015): *Bounded Synthesis for Petri Games*. In Roland Meyer, André Platzer & Heike Wehrheim, editors: *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Proceedings*, *Lecture Notes in Computer Science* 9360, Springer, pp. 223–237, doi:10.1007/978-3-319-23506-6_15.

[7] Bernd Finkbeiner & Paul Gölz (2018): *Synthesis in Distributed Environments*. In Satya Lokam & R. Ramanujam, editors: *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 93, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, pp. 28:1–28:14, doi:10.4230/LIPIcs.FSTTCS.2017.28.

[8] Bernd Finkbeiner & Ernst-Rüdiger Olderog (2017): *Petri games: Synthesis of distributed systems with causal memory*. *Information and Computation* 253, pp. 181–203, doi:10.1016/j.ic.2016.07.006. GandALF 2014.

[9] Bernd Finkbeiner & Sven Schewe (2005): *Uniform Distributed Synthesis*. In: *20th IEEE Symposium on Logic in Computer Science (LICS 2005), Proceedings*, pp. 321–330, doi:10.1109/LICS.2005.53.

[10] Blaise Genest, Hugo Gimbert, Anca Muscholl & Igor Walukiewicz (2013): *Asynchronous Games over Tree Architectures*. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska & David Peleg, editors: *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Proceedings, Part II*, *Lecture Notes in Computer Science* 7966, Springer, pp. 275–286, doi:10.1007/978-3-642-39212-2_26.

[11] Hugo Gimbert (2017): *On the Control of Asynchronous Automata*. In Satya V. Lokam & R. Ramanujam, editors: *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, India*, *LIPIcs* 93, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 30:1–30:15, doi:10.4230/LIPIcs.FSTTCS.2017.30.

[12] Hugo Gimbert (2022): *Distributed Asynchronous Games With Causal Memory are Undecidable*. *Log. Methods Comput. Sci.* 18(3), doi:10.46298/lmcs-18(3:30)2022.

[13] Paul Hannibal & Ernst-Rüdiger Olderog (2022): *The Synthesis Problem for Repeatedly Communicating Petri Games*. In Luca Bernardinello & Laure Petrucci, editors: *Application and Theory of Petri Nets and Concurrency - 43rd International Conference, PETRI NETS 2022, Proceedings*, *Lecture Notes in Computer Science* 13288, Springer, pp. 236–257, doi:10.1007/978-3-031-06653-5_13.

[14] O. Kupferman & M.Y. Vardi (2001): *Synthesizing distributed systems*. *Proceedings - Symposium on Logic in Computer Science*, pp. 389–398, doi:10.1109/LICS.2001.932514.

[15] P. Madhusudan & P. S. Thiagarajan (2002): *A Decidable Class of Asynchronous Distributed Controllers*. In Lubos Brim, Petr Jancar, Mojmír Kretínský & Antonín Kucera, editors: *CONCUR 2002 - Concurrency Theory, 13th International Conference, Proceedings*, *Lecture Notes in Computer Science* 2421, Springer, pp. 145–160, doi:10.1007/3-540-45694-5_11.

[16] P. Madhusudan, P. S. Thiagarajan & Shaofa Yang (2005): *The MSO Theory of Connectedly Communicating Processes*. In Ramaswamy Ramanujam & Sandeep Sen, editors: *FSTTCS 2005: Foundations of Software*

Technology and Theoretical Computer Science, 25th International Conference, Proceedings, *Lecture Notes in Computer Science* 3821, Springer, pp. 201–212, doi:`10.1007/11590156_16`.

[17] Anca Muscholl (2015): *Automated Synthesis of Distributed Controllers*. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi & Bettina Speckmann, editors: *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Proceedings, Part II*, *Lecture Notes in Computer Science* 9135, Springer, pp. 11–27, doi:`10.1007/978-3-662-47666-6_2`.

[18] Amir Pnueli & Roni Rosner (1989): *On the Synthesis of an Asynchronous Reactive Module*. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini & Simona Ronchi Della Rocca, editors: *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Proceedings*, *Lecture Notes in Computer Science* 372, Springer, pp. 652–671, doi:`10.1007/BFb0035790`.

[19] Amir Pnueli & Roni Rosner (1990): *Distributed Reactive Systems Are Hard to Synthesize*. In: *31st Annual Symposium on Foundations of Computer Science, Volume II*, pp. 746–757, doi:`10.1109/FSCS.1990.89597`.

[20] R. Rosner (1992): *Modular synthesis of reactive systems*. Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel.

Counterfactual Causality for Reachability and Safety based on Distance Functions

Julie Parreaux
Aix Marseille Univ, CNRS, LIS, Marseille, France
julie.parreaux@univ-amu.fr

Jakob Piribauer
Technische Universität Dresden, Germany
Technische Universität München, Germany
jakob.piribauer@tu-dresden.de

Christel Baier
Technische Universität Dresden, Germany
christel.baier@tu-dresden.de

Investigations of causality in operational systems aim at providing human-understandable explanations of *why* a system behaves as it does. There is, in particular, a demand to explain what went wrong on a given counterexample execution that shows that a system does not satisfy a given specification. To this end, this paper investigates a notion of counterfactual causality in transition systems based on Stalnaker's and Lewis' semantics of counterfactuals in terms of most similar possible worlds and introduces a novel corresponding notion of counterfactual causality in two-player games. Using distance functions between paths in transition systems to capture the similarity of executions, this notion defines whether reaching a certain set of states is a cause for the fact that a given execution of a system satisfies an undesirable reachability or safety property. Similarly, using distance functions between memoryless strategies in reachability and safety games, it is defined whether reaching a set of states is a cause for the fact that a given strategy for the player under investigation is losing.

The contribution of the paper is two-fold: In transition systems, it is shown that counterfactual causality can be checked in polynomial time for three prominent distance functions between paths. In two-player games, the introduced notion of counterfactual causality is shown to be checkable in polynomial time for two natural distance functions between memoryless strategies. Further, a notion of explanation that can be extracted from a counterfactual cause and that pinpoints changes to be made to the given strategy in order to transform it into a winning strategy is defined. For the two distance functions under consideration, the problem to decide whether such an explanation imposes only minimal necessary changes to the given strategy with respect to the used distance function turns out to be coNP-complete and not to be solvable in polynomial time if P is not equal to NP, respectively.

1 Introduction

Modern software and hardware systems have reached a level of complexity that makes it impossible for humans to assess whether a system behaves as intended without tools tailored for this task. To tackle this problem, automated verification techniques have been developed. *Model checking* is one prominent such technique: A model-checking algorithm takes a mathematical model of the system under investigation and a formal specification of the intended behavior and determines whether all possible executions of the model satisfy the specification. While the results of a model-checking algorithm provide guarantees on

Funding: This work was partly funded by DFG Grant 389792660 as part of TRR 248 (Foundations of Perspicuous Software Systems), the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), and the DFG projects BA-1679/11-1 and BA-1679/12-1, and the ANR project Ticktac (ANR-18-CE40-0015).

the correctness of a system or affirm the presence of an error, their usefulness is, nevertheless, limited as they do not provide a human-understandable explanation of the behavior of the system.

To provide additional information on *why* the system behaves as it does, certificates witnessing the result of the model-checking procedure, in particular counterexample traces in case of a negative result, have been studied extensively (see, e.g., [10, 29, 9, 30]). Due to the potentially still enormous size of counterexample traces and other certificates, a line of research has emerged that tries to distill comprehensible explications of what causes the system to behave as it does using formalizations of *causality* (see, e.g., [32, 33, 2]).

Forward- and backward-looking causality There are two fundamentally different types of notions of causality: *forward-looking* and *backward-looking* notions [34]. In the context of operational system models, forward-looking causality describes general causal relations between events that might happen along some possible executions. Backward-looking causality, on the other hand, addresses the causal relation between events along a given execution of the system model. This distinction is captured in more general contexts by the distinction between *type-level* causality addressing general causal dependencies between events that might happen when looking forward in a world model, and *token-level* or *actual* causality, corresponding to the backward view, that addresses causes for a particular event that actually happened (see, e.g., [16]).

Notions of *necessary* causality are typically forward-looking: A necessary cause C for an effect E is an event that occurs on every execution that exhibits the effect E (see, e.g., [3], and for a philosophical analysis of necessity in causes [28]). The backward view naturally arises when the task is to explain what went wrong after an undesired effect has been observed. In the verification context, the backward view is natural for explaining counterexamples, see e.g. [42, 6, 15, 35, 38, 39]. Most of these techniques rely on the *counterfactuality* principle, which has been originally studied in philosophy [20, 21, 37, 26, 27] and formalized mathematically by Halpern and Pearl [17, 18, 19, 16]. Intuitively, counterfactual causality requires that the effect would not have happened, if the cause had not occurred, in combination with some minimality constraints for causes. The most prominent account for the semantics of the involved counterfactual implication is provided by Stalnaker and Lewis [37, 26, 27] in terms of closest, i.e., most similar, possible worlds. The statement "if the cause C had not occurred, then the effect E would not have occurred" holds true if in the worlds that are most similar to the actual world and in which C did not occur, E also did not occur. Interpreting executions of a system as possible worlds, the actual world is an execution π where both the effect E and its counterfactual cause C occur, while the effect E does not occur in alternative executions that are as similar as possible to π and that do not exhibit C.

For a more detailed discussion on the distinction between forward- and backward looking causality and related concepts for responsibility, we refer the reader, e.g., to [34, 40, 41, 4, 2].

Defining counterfactual causality in transition systems and reachability games To define our backward-looking notion of counterfactual causality in transition systems, we follow an approach similar to the one by Groce et al [14] who presented a Stalnaker-Lewis-style formalization of counterfactual dependence of events using distance functions. We consider the case where effects are reachability or safety properties and causes are sets of states. To illustrate the idea, let $\mathscr{T}$ be a transition system and let E and C be disjoint sets of states of $\mathscr{T}$ indicating a reachability effect and a potential cause, respectively. Consider an execution π that reaches the effect set and the potential cause set. We employ the counterfactual reading of causality by Stalnaker and Lewis by viewing executions as possible worlds using a similarity metric d on paths: Reaching C was a cause for π to reach E if all paths ζ, that do not

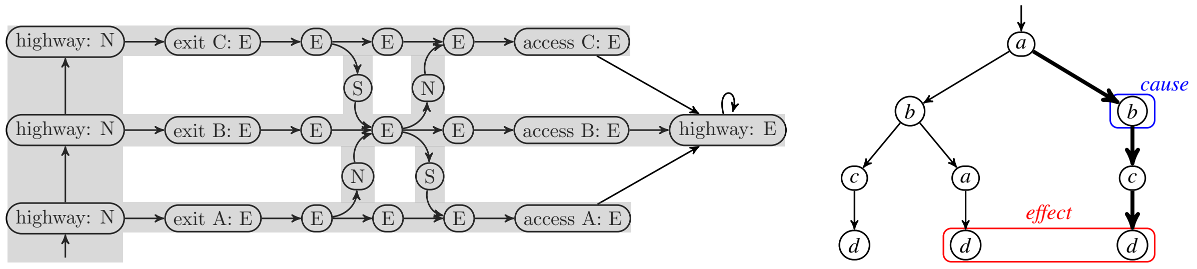


Figure 1: On the left: example transition system modelling a traffic grid (Ex. 1). On the right: Example of a d_{Hamm}-counterfactual cause that is not a d_{pref}-counterfactual cause (Ex. 3).

reach C and that are most similar to π according to d among all paths with this property, satisfy $\zeta \models \Box\neg E$, i.e., they do not reach E. So, we first determine the minimal similarity-distance $d_{\min} = \min\{d(\pi,\zeta) \mid \zeta \models \Box\neg C\}$ from π to a path ζ that does not reach C. Then, we check whether all paths that do not reach C and have similarity-distance $d_{\min}$ to π do not reach E: Do all $\zeta \in \{\zeta' \mid d(\pi,\zeta') = d_{\min}$ and $\zeta' \models \Box\neg C\}$ satisfy $\Box\neg E$? If the answer is yes, it is the case that "if C had not occurred, then E would not have occurred" and so C is a counterfactual cause for E on π.

Example 1. Consider the following distance function on paths in a labeled transition system $\mathscr{T}$ with states S and a labeling function $L\colon\ S \to \mathscr{A}$ for a set of labels $\mathscr{A}$: For paths $\pi = s_0, s_1, \ldots$ and $\pi' = t_0, t_1, \ldots$, we define $dist(\pi,\pi') = |\{n \in \mathbb{N} \mid L(s_n) \neq L(t_n)\}|$. So, paths are more similar if their traces differ at fewer positions. To determine whether C is a cause for E on π, we first determine what the least number $n_{\min}$ of changes to the state labels of π is to obtain a path ζ that does not reach C. Then, we have to check whether all paths differing from π in $n_{\min}$ labels and not reaching C do not reach E. If this is the case, C is a counterfactual cause for E on π with respect to *dist*.

Now, consider the example transition system $\mathscr{T}$ modelling a road system with a highway going north that has three exits into a small town which can be left again on a highway heading east depicted in Fig. 1. Each state is labeled with N, E, or S for north, east, and south as indicated in Fig. 1 depending on the direction the cars move on the respective road. Say, an agent traverses the system via the path π with trace NNE^{ω}, i.e., by taking exit B from the first highway and then going eastwards straight through the town. Assume that there is a traffic jam on access B while the other access roads are free. The question is now whether taking exit B was a cause for being stuck in slow traffic later on, i.e., for the effect $\{$access B$\}$. First, note here that the set $\{$exit B$\}$ is not a forward-looking necessary cause for reaching $\{$access B$\}$. There are paths through the system that avoid $\{$exit B$\}$, but reach $\{$access B$\}$.

However, given the fact that the agent traversed the town by going straight eastwards, it is reasonable to say that the agent would have reached a different access road if she had taken a different exit from the first highway. This is reflected in the counterfactual definition using *dist*: There are two paths that do not reach exit B and whose traces differ from π at only one position, namely the paths with trace NE^{ω} and $NNNE^{\omega}$. These paths do also not reach access B. So, $\{$exit B$\}$ is a counterfactual cause for $\{$access B$\}$ on π; if the agent had taken exit A or C, she would not have hit the low traffic flow at access B. ⌟

In the context of two-player reachability games, causality has been used as a tool to solve games [1]. In our work, we focus on explaining why a certain strategy does not allow the player to win. More precisely, in reachability games between players with a safety and the complementing reachability objective, respectively, we consider the situation where one of the players Π has a winning strategy, but loses the game using a strategy σ. We introduce a notion of counterfactual causality that aims to provide insights into what is wrong with strategy σ by transferring the counterfactual definition using distance functions

distance d	causality
prefix	in P (Thm. 4)
Hamming	in P (Thm. 5)
Levenshtein	in P (Thm. 8)

distance d	causality	explanations
Hausdorff lifting d^H_{pref} of the prefix distance	in P (Thm. 12)	
Hamming strategy distance d^s_{Hamm}	in P in acyclic games (Thm. 13)	coNP-complete (Cor. 21)
Hausdorff-inspired distance d^*		not in P if P$\neq$NP (Cor. 21)

Table 1: Overview of the complexity results. On the left, the complexities of checking d-counterfactual causality in transition systems, and on the right, the complexities of checking d-counterfactual causality and d-minimality of explanations in reachability games.

d on memoryless strategies. A set of states C is said to be a d-counterfactual cause for the fact that σ is losing if all memoryless strategies τ, that make sure that C is not reached and have minimal d-distance to σ among all such strategies, are winning. Furthermore, we introduce *counterfactual explanations* that specify minimally invasive changes of σ's decisions required to turn σ into a winning strategy.

Contributions

- We show that d-counterfactual causal relationships in transition systems (defined as in [14]) can be checked in polynomial time for the following three distance metrics d (Sec. 3.2):
  1. the prefix distance: paths are more similar if their traces share a longer prefix.
  2. the Hamming distance that counts the positions at which traces of paths differ.
  3. the Levenshtein distance that counts how many insertions, deletions, and substitutions are necessary to transform the trace of one path to the trace of another path.

  Furthermore, we show that the notion of d-counterfactual causality for the Hamming distance is consistent with Halpern and Pearl's but-for causes [18, 19].
- In reachability games, we provide a generalization of this notion using similarity metrics on memoryless deterministic strategies. We show that for the Hausdorff lifting of the prefix distance on paths to a distance function on memoryless deterministic strategies, the resulting notion can be checked in polynomial time (Sec. 4.1).
- We introduce a notion of *counterfactual explanation* that can be computed from a counterfactual cause (Sec. 4.2). An explanation specifies where a non-winning strategy needs to be changed. Of particular interest are D-minimal explanations that enforce only minimal necessary changes with respect to a distance function D on strategies. For two distance functions related to the Hamming distance, we show that checking whether an explanation is minimal is coNP-complete and not in P if P$\neq$NP, respectively.

An overview of the complexity results can be found in Table 1.

Related work Ways to pinpoint the problematic steps in a counterexample trace by localizing errors have widely been studied [42, 6, 15, 35, 38, 39]. For counterfactuality in transition systems, we follow the approach of [14] with distance metrics. In contrast to the causes in this paper, causes in [14] are formulas in an expressive logic that can precisely talk about the valuation of variables after a certain number of steps. Further [14] is not concerned with checking causality, but with finding causes, which, due to the expressive type of causes, algorithmically boils down to finding executions avoiding the effect with a minimal distance to the given one.

Based on counterfactuality, Halpern and Pearl [18, 19, 16] provided an influential formalization of causality using structural equation models, which has served as the basis for various notions of causality in the verification context (see,e.g., [7, 24]). A key ingredient is the notion of *intervention* to provide a semantics for the counterfactual implication in Hume's definition of causality. An intervention in a structural equation model sets a variable to a certain value by force, ignoring its dependencies on other variables, and evaluates the effects of this enforced change. In a sense, a minimal set of interventions to avoid a cause then leads to a most similar execution avoiding the cause. We will discuss the relations between our definition and the Halpern-Pearl definition in more detail in Section 3.3. In [7], interventions are employed to counterexample traces in transition systems by allowing to flip atomic propositions along a trace. In contrast to our notion of counterfactual causes, this is tailored for complex linear time properties, but does not provide insights for reachability and safety. Furthermore, the flipping of atomic propositions can be seen as a change in the transition system while our definition considers alternative executions without manipulating the system. In [11], the Halpern-Pearl approach is applied to provide a counterfactual definition of causality in reactive systems. A distance partial order, namely the subset relation on sets of positions at which traces differ, is used to describe which interventions are acceptable as they constitute minimal changes necessary to avoid the cause. Checking causality is shown to be decidable by a formulation as a hyperlogic model-checking problem. Furthermore, notions of necessary and sufficient causes as sets of states in transition systems have been considered [3]. These do not rely on the counterfactuality principle and are of forward-looking nature.

We are not aware of formalisations of causality in game structures. The related concept of responsibility, has been investigated in multi-agent models [40, 41]. Notions of forward and backward responsibility of players in multi-player game structures with acyclic tree-like arena have been studied [4].

For a detailed overview of work on causality and related concepts in operational models, we refer the reader to the survey articles [8, 2].

2 Preliminaries

We briefly present notions we use and our notation. For details, see [5, 13].
Transition systems. A transition system is a tuple $\mathscr{T} = (S, s_{init}, \to, L)$ where S is a finite set of states, $s_{init} \in S$ is an initial state, $\to \subseteq S \times S$ is a transition relation and $L\colon S \to 2^{\mathsf{AP}}$ is a labeling function where AP is a set of atomic propositions. A path in a transition system is a finite or infinite sequence of states $s_0 s_1 \dots$ such that $s_0 = s_{init}$ and, for all suitable indices i, there is a transition from s_i to s_{i+1}, i.e., $(s_i, s_{i+1}) \in \to$. Given a path $\pi = s_0 s_1 \dots$, we denote its trace $L(s_0)L(s_1)\dots$ by $L(\pi)$. If there are no outgoing transitions from a state, we call the state *terminal*.
Computation tree logic (CTL). The branching-time logic CTL consists of state formulas that are evaluated at states in a transition system formed by $\Phi ::= \top \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$ where $a \in \mathsf{AP}$ is an atomic proposition and path formulas evaluated on paths formed by $\varphi ::= \bigcirc\Phi \mid \Phi \mathrm{U} \Phi$. The semantics of the temporal operators in path formulas is as usual. We use the abbreviations $\Diamond\Phi$ for $\top \mathrm{U} \Phi$ and $\Box\Phi = \neg\Diamond\neg\Phi$ and also allow sets of states T in the place of state formulas. The semantics of $\exists\varphi$ are that there exists a path starting in the state at which the formula is evaluated that satisfies φ; $\forall\varphi$ is defined dually to that as usual. Model checking of CTL-formulas can be done in polynomial time. For details, see [5].
Reachability games. A *reachability game* is a tuple $\mathscr{G} = (V, v_{\mathsf{i}}, \Delta)$ where $V = V_{\mathsf{Reach}} \uplus V_{\mathsf{Safe}} \uplus V_{\mathit{Eff}}$ is the set of vertices shared between players Reach and Safe, and some target vertices V_{Eff} (*Eff* for effect).

$v_{\mathsf{i}} \in V \setminus V_{\mathit{Eff}}$ is the initial vertex and $\Delta \subseteq V \times V$ is the set of edges. We denote by $\Delta(v)$ the set of edges from v. W.l.o.g., we assume that target vertices are terminal states, i.e. for all vertices $v \in V_{\mathit{Eff}}$, $\Delta(v) = \emptyset$. A *finite play* is a finite sequence of vertices $\pi = v_0 v_1 \cdots v_k \in V^*$ such that for all $0 \leq i < k$, $(v_i, v_{i+1}) \in \Delta$. A *play* is either a finite play ending in a target vertex, or an infinite sequence of vertices such that every finite prefix is a finite play. Transition systems can be viewed as one-player games.

A *strategy* for Reach in a reachability game $\mathscr{G}$ is a mapping $\sigma\colon V^* V_{\mathsf{Reach}} \to V$. A play or finite play $\pi = v_0 v_1 \cdots$ is a σ*-play* if for all k with $v_k \in V_{\mathsf{Reach}}$, we have $\sigma(v_0 \cdots v_k) = v_{k+1}$. A strategy σ is an *MD-strategy* (for memoryless deterministic) if for all finite plays ξ and ξ' with the same last vertex, we have that $\sigma(\xi) = \sigma(\xi')$. In this paper, we mainly use MD-strategies and write $\sigma(v_k)$ instead of $\sigma(v_0 \cdots v_k)$ for MD-strategies σ. Moreover, under a (partial) MD-strategy σ, we define the *reachability game under* σ, denoted by $\mathscr{G}^\sigma = (V, v_{\mathsf{i}}, \Delta^\sigma)$, by removing edges not chosen by σ, i.e., $\Delta^\sigma = \Delta \setminus \{(v,v') \in \Delta \mid v \in V_{\mathsf{Reach}}$ and $\sigma(v)$ is defined and $\sigma(v) \neq (v,v')\}$. When σ is completely defined, $\mathscr{G}^\sigma$ is a transition system. Finally, a strategy is *winning* if all σ-plays starting in v_{i} end in a target vertex. Analogous definitions apply to Safe. In reachability games, either Reach or Safe wins with an MD-strategy. This winning strategy can be computed in polynomial time (see, e.g., [13]).

Distance function. A *distance function* on a set A is a function $d\colon A \times A \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ such that $d(x,x) = 0$ for all $x \in A$ and $d(x,y) = d(y,x)$ for all $x, y \in A$. It is called a *pseudo-metric* if additionally $d(x,y) + d(y,z) \geq d(x,z)$ for all $x,y,z \in A$, and a *metric* if further $d(x,y) = 0$ holds iff $x = y$ for all $x, y \in A$.

3 Counterfactual causes in transition systems

In this section, we introduce the backward-looking notion of counterfactual causes in transition systems using distance functions (Section 3.1). Afterwards, we prove that the definition can be checked in polynomial time for three well-known distance functions (Section 3.2). Finally, we illustrate similarities between our notion of counterfactual causality to the definition of causality by Halpern and Pearl (Sec 3.3). Proofs omitted here can be found in the extended version [31].

3.1 Definition

The effects we consider are reachability or safety properties $\Phi = \lozenge E$ or $\Phi = \square \neg E$ for a set of states E. As the behavior of the system after E has been seen is not relevant for these properties, we assume that E consists of terminal states.

Definition 2 (d-counterfactual cause in transition systems)**.** Let $\mathscr{T}$ be a transition system and let d be a distance function on the set of maximal paths of $\mathscr{T}$. Let E be a set of terminal states and let C be a set of states disjoint from E. Let $\Phi = \lozenge E$ or $\Phi = \square \neg E$. Given a maximal path π that visits C and satisfies Φ in $\mathscr{T}$, we say that C is a *d-counterfactual cause for* Φ *on* π if

1. there is a maximal path ρ in $\mathscr{T}$ that does not visit C, and
2. all maximal paths ρ with $\rho \vDash \square \neg C$ with minimal distance to π do not satisfy Φ. In other words, all maximal paths ρ with $\rho \vDash \square \neg C$ such that $d(\pi, \rho) \leq d(\pi, \rho')$ for all ρ' with $\rho' \vDash \square \neg C$ satisfy $\rho \vDash \neg \Phi$.

The choice of the similarity distance d of course heavily influences the notion of d-counterfactual cause. In this paper, we will instantiate the definition with three distance functions that are among the most prominent distance functions between traces (or words). An experimental investigation to clarify in which situations what kind of distance functions leads to a desirable notion of causality, however, remains as future work.

Prefix metrics d_{pref}^{AP} and d_{pref}: given two paths π and ρ, let $n(\pi,\rho)$ be the length of the longest common prefix of their traces $L(\pi)$ and $L(\rho)$. Then, $d_{pref}^{\mathsf{AP}}(\pi,\rho) \overset{\text{def}}{=} 2^{-n(\pi,\rho)}$. We can also define the distance on paths instead of traces, which will be used later on: $d_{pref}(\pi,\rho) \overset{\text{def}}{=} 2^{-m(\pi,\rho)}$ where $m(\pi,\rho)$ is the length of the longest common prefix of π and ρ as paths. This can be seen as a special case of d_{pref}^{AP} if we assume that all states have a unique label.

The prefix metric measures similarity in a temporal way saying that executions are more similar if they initially agree for a longer period of time. If no further structure of the transition system or meaning of the labels is known, this distance function might be a reasonable choice for counterfactual causality.

Hamming distance d_{Hamm}: Given two words $w = w_0 \dots w_n$ and $v = v_0 \dots v_n$ of the same length, we define $d_{Hamm}(w,v) \overset{\text{def}}{=} |\{0 \le i \le n \mid w_i \neq v_i\}|$. For two maximal paths π and ρ of the same length in a transition system $\mathscr{T}$ with labeling function L, we define $d_{Hamm}(\pi,\rho) \overset{\text{def}}{=} d_{Hamm}(L(\pi),L(\rho))$. So, the distance between two paths is the Hamming distance of their traces.

The Hamming distance seems to be a reasonable measure if a system naturally proceeds through different layers, e.g., if a counter is increased in each step. Then, traces are viewed to be more similar if they agree on more layers. The temporal order of these layers, however, does not play a role.

Levenshtein distance d_{Lev} [25]: Given two words $w = w_0 \dots w_n$ and $v = v_0 \dots v_m$, the Levenshtein distance is defined as the minimal number of editing operations needed to produce v from w where the allowed operations are insertion of a letter, deletion of a letter, and substitution of a letter by a different letter. Formally, we define d_{Lev} in terms of *edit sequences*. Let Σ be an alphabet and $v,w \in \Sigma^* \cup \Sigma^\omega$ be two words over Σ. The *edit alphabet* for Σ is defined as $\Gamma \overset{\text{def}}{=} (\Sigma \cup \{\varepsilon\})^2 \setminus \{(\varepsilon,\varepsilon)\}$ where ε is a fresh symbol. An edit sequence for v and w is now a word $\gamma \in \Gamma^* \cup \Gamma^\omega$ such that the projection of γ onto the first component results in v when all εs are removed and the projection of γ onto the second component results in w when all εs are removed. E.g., let $\Sigma = \{a,b,c\}$, $v = abbc$ and $w = accbc$. One edit sequence is $\gamma = (a,a)(b,c)(\varepsilon,c)(b,b)(c,c)$. The weight of an edit sequence $\gamma = \gamma_1\gamma_2\dots$ is defined as $wgt(\gamma) = |\{i \mid \gamma_i \neq (\sigma,\sigma) \text{ for all } \sigma \in \Sigma\}|$. Then, for all words $v \in \Sigma^* \cup \Sigma^\omega$ and $w \in \Sigma^* \cup \Sigma^\omega$, we define $d_{Lev}(v,w) = \min\{wgt(\gamma) \mid \gamma \text{ is an edit sequence for } v \text{ and } w\}$. Again, we obtain a pseudo-metric on paths via the Levenshtein metric on traces.

The Levenshtein distance might be particularly useful if labels model actions that are taken. Two executions that are obtained by sequences of actions that only differ by inserting or leaving out some actions, but otherwise using the same actions, are considered to be similar in this case.

Example 3. Let us illustrate counterfactual causality for the prefix metric d_{pref} and the Hamming distance d_{Hamm}. Consider the transition system depicted in Figure 1. A path π as indicated by the bold arrows on the right via the potential cause to the effect has been taken: This is not a d_{pref}-counterfactual cause on π: The most similar paths to π that do not reach *cause* are both paths that move to the left initially. As one of these paths reaches *effect*, the set *cause* is not a d_{pref}-counterfactual cause for reaching *effect*.

Considering the distance function d_{Hamm} with the labels of the states as in Figure 1, we get a different result: The trace of π is $abcd$. The paths that avoid the potential cause have traces $abcd$ and $abad$, respectively. So, the most similar path avoiding *cause* is the path on the left with trace $abcd$ that also avoids *effect*. So, *cause* is a d_{Hamm}-counterfactual cause on π for $\lozenge$*effect*. Intuitively, this can be understood as saying if the system had avoided *cause* but otherwise behaved (as similar as possible to) as it did in terms of the produced trace, the effect would not have occurred. In particular, if labels represent actions that have been chosen, this is a reasonable reading of causality. ⌟

3.2 Checking counterfactual causality in transition systems

In this section, we provide algorithms to check d-counterfactual causality for the three distance functions d^{AP}_{pref}, d_{Hamm}, and d_{Lev}. For these algorithms, a maximal execution π of the system has to be given. We assume that π is a finite path ending in a terminal state. The problem to find causes that are small or satisfy other desirable properties is not addressed in this paper and remains as future work. We will briefly come back to this in the conclusions.

Prefix distance. First, we consider d^{AP}_{pref}-counterfactual causality and hence d_{pref}-counterfactual causality as a special case.

Theorem 4. *Let $\mathcal{T} = (S, s_{init}, \to, L)$ be a transition system, E a set of terminal states, C a set of states disjoint from E, and $\Phi = \lozenge E$ or $\Phi = \square\neg E$. Let $\pi = s_0 \dots s_n$ be an execution reaching C and satisfying Φ. It is decidable in polynomial time whether C is a d^{AP}_{pref}-counterfactual cause for Φ on π.*

Proof sketch. The following algorithm solves the problem in polynomial time: First, we determine the last index i s.t. C is not reached on any path with trace $L(s_0), \dots, L(s_i)$ and s.t. C is avoidable from some state that is reachable via a path with trace $L(s_0), \dots, L(s_i)$. In order to that, we recursively construct sets T_{j+1} of states that are reachable via paths with trace $L(s_0), \dots, L(s_{j+1})$ and check for all states $t \in T_{j+1}$ whether $t \models \exists\square\neg C$. If no such state exists, we have found the first index $j+1$ such that C is not avoidable anymore after trace $L(s_0), \dots, L(s_{j+1})$; so we have found $i = j$. Now, we check whether $t \models \forall(\Phi \to \lozenge C)$ for all $t \in T_i$. If this is the case, C is a d^{AP}_{pref}-counterfactual cause for E on π; otherwise, it is not. $\square$

Hamming distance. The Hamming distance is only defined for words of the same length. We will hence first consider only transition systems in which all maximal paths have the same length. We can think of such transition systems as being structured in layers with indices 1 to k for some k. Transitions can then only move from a state on layer $i < k$ to a state on layer $i+1$. Afterwards, we consider a simple generalization of the Hamming distance to words of different lengths.
Original Hamming distance. Let $\mathcal{T} = (S, s_{init}, \to, L)$ be a transition system in which all maximal paths have the same length k. We annotate all states with the layer they are on: For each state $s \in S$, there is a unique length $n \leq k$ of all paths from s_{init} to s. We will say that state s lies on layer n in this case. By our assumption that effect states are terminal, the states E are all located on the last layer k. We assume furthermore that all effect states have the same labels.

Theorem 5. *Let $\mathcal{T} = (S, s_{init}, \to, L)$ be a transition system in which all maximal paths have the same length k. Let E be a set of terminal states and let $C \subseteq S$ be a set of states disjoint from E. Let $\Phi = \lozenge E$ or $\Phi = \square\neg E$. Let $\pi = s_0 \dots s_n$ be an execution reaching C and satisfying Φ. It is decidable in polynomial time whether C is a d_{Hamm}-counterfactual cause for Φ on π.*

Proof sketch. We sketch the proof for the case that $\Phi = \lozenge E$. We equip the states in S with a weight function $wgt\colon S \to \{0,1\}$ such that the d_{Hamm}-distance of a path to π is equal to the accumulated weight of that path. A state t on layer i gets weight 1 if its label is different to $L(s_i)$. Otherwise, it gets weight 0. Now, we can check whether C is a d_{Hamm}-counterfactual cause, as follows: We remove all states in C and compute a shortest (i.e., weight-minimal) path ζ to E and a shortest path ξ to any terminal state. If the weight of ξ is lower than the weight of ζ, the paths avoiding C that are d_{Hamm}-closest to π do not reach E and C is a d_{Hamm}-counterfactual cause for $\lozenge E$ on π; otherwise, it is not. $\square$

Remark 6. The Hamming distance between paths could easily be extended to account for different levels of similarities between labels: Given a similarity metric d on the set of labels, one could define the distance between two paths $\pi = s_1 \dots s_k$ and $\rho = t_1 \dots t_k$ as $d'_{Hamm}(\pi, \rho) \overset{\text{def}}{=} \sum_{i=1}^{k} d(s_i, t_i)$. The algorithm in the proof of Theorem 5 can now easily be adapted to this modified Hamming distance by defining the weight function on the transition system in the obvious way.

Generalized Hamming distance. The assumption in the previous section that all paths in a transition system have the same length is quite restrictive. Hence, we now consider the following generalized version d_{gHamm} of the Hamming distance: For words $w = w_1 \dots w_n$ and $v = v_1 \dots v_m$, we define

$$d_{gHamm}(w,v) \overset{\text{def}}{=} \begin{cases} d_{Hamm}(w, v_{[1:n]}) + (m-n) & \text{if } n \leq m, \\ d_{Hamm}(w_{[1:m]}, v) + (n-m) & \text{otherwise.} \end{cases}$$

So d_{gHamm} takes a prefix of the longer word of the same length as the shorter word, computes the Hamming distance of the prefix and the shorter word, and adds the difference in length of the two words.

Theorem 7. *Let $\mathscr{T} = (S, s_{init}, \rightarrow, L)$ be a transition system, E a set of terminal states, and C a set of states disjoint from E. Let $\Phi = \Diamond E$ or $\Phi = \Box\neg E$. Let $\pi = s_0 \dots s_n$ be an execution reaching C and satisfying Φ. It is decidable in polynomial time whether C is a d_{gHamm}-counterfactual cause for Φ on π.*

Proof sketch. We adapt the proof of Theorem 5: We take $|\pi|$-many copies of the state space S an let transitions lead from one copy to the next. In the ith copy states with the same label as s_i get weight 0 and all other states get weight 1. Furthermore, we add transitions with weight $|\pi| - i$ from terminal states in a copy $i < |\pi|$ to the same state in the last copy to account for path that are shorter than π. The weight $|\pi| - i$ corresponds to the value added in the generalized Hamming distance when paths of different length are compared. To account for paths longer than π, we furthermore allow transitions with weight 1 within the last copy. These transitions are then taken until a terminal state is reached. With these adaptations, the proof can be carried out analogously to the proof of Theorem 5. □

Levenshtein distance. The idea to check d_{Lev}-counterfactual causality is to construct a weighted transition system to check causality via the computation of shortest paths as for the Hamming distance. So, let $\mathscr{T} = (S, \rightarrow, s_{init}, L)$ be a transition system labeled by L with symbols from $\Sigma = 2^{\mathsf{AP}}$. Let E be a set of terminal states and C a set of states disjoint from E. Let $\Phi = \Diamond E$ or $\Phi = \Box\neg E$. Let $\pi = s_1 \dots s_n$ be a maximal path reaching C and satisfying Φ. The transition system we construct contains transitions corresponding directly to the edit operations insertion, deletion and substitution. A path in the constructed transition system then corresponds to an edit sequence between the trace of π and the trace of another path in $\mathscr{T}$. This construction shares some similarities with the construction of Levenshtein automata [36] that accept all words with a Levenshtein distance below a given constant c from a fixed word w.

Now, we formally construct the new weighted transition system $\mathscr{T}^{\pi}_{d_{Lev}}$: The state space of this transition system is $S \times \{1, \dots, n\}$ with the initial state $(s_{init}, 1)$. The labeling function is not used. In $\mathscr{T}^{\pi}_{d_{Lev}}$, we allow the following transitions labelled with letters from the edit alphabet Γ:

1. a transition from (s,i) to $(t, i+1)$ labeled with $(L(s_{i+1}), L(t))$ for each $(s,t) \in \rightarrow$ and $i < n$,
2. a transition from (s,i) to (t,i) labeled with $(\varepsilon, L(t))$ for each $(s,t) \in \rightarrow$ and $i \leq n$,
3. a transition from (s,i) to $(s, i+1)$ labeled with $(L(s_{i+1}), \varepsilon)$ for each $s \in S$ and $i < n$.

Note that the terminal states in $\mathscr{T}^{\pi}_{d_{Lev}}$ are all contained in $S \times \{n\}$. Any maximal path in $\mathscr{T}^{\pi}_{d_{Lev}}$ corresponds to a maximal path ρ in $\mathscr{T}$. This path ρ is obtained by moving from a state s to a state t in $\mathscr{T}$ whenever a corresponding transition of type 1 or 2 is taken in $\mathscr{T}^{\pi}_{d_{Lev}}$. Transitions of type 3 do not correspond to a step in $\mathscr{T}$ and stay in the same state.

Furthermore, given a finite path τ in $\mathscr{T}^{\pi}_{d_{Lev}}$ and the corresponding path $\rho = t_1 \dots t_k$ in $\mathscr{T}$, the labels of the transitions of τ form an edit sequence for the words $L(s_2)\dots L(s_n)$ and $L(t_2)\dots L(t_k)$. To see this, observe that, for each $i > 1$, whenever the copy $S \times \{i\}$ is entered in $\mathscr{T}^{\pi}_{d_{Lev}}$, the label of the transition contains $L(s_i)$ in the first component; if a transition stays in a copy $S \times \{i\}$, the label contains ε in the first component. So, the projection onto the first component of the labels of the transitions of τ is indeed $L(s_2)\dots L(s_n)$, potentially with εs in between. In the second component, whenever a transition of type 1 or 2 is taken, the label is simply the label of the corresponding state in ρ. Transitions of type 3 have ε in the second component of their label. Note here that ρ and π both start in s_{init} and that we could hence add $(L(s_{init}), L(s_{init}))$ to the beginning of the edit sequence to obtain an edit sequence for the full traces of τ and ρ. Note that also for infinite paths $\tau = t_1t_2\dots$ in $\mathscr{T}^{\pi}_{d_{Lev}}$ the transition labels provide an edit sequence for the words $L(s_2)\dots L(s_n)$ and $L(t_2)L(t_3)\dots$. Vice versa, a finite maximal path $\rho = t_1 \dots t_k$ in $\mathscr{T}$ together with an edit sequence γ for $L(s_2)\dots L(s_n)$ and $L(t_2)\dots L(t_k)$ provides a maximal path τ in $\mathscr{T}^{\pi}_{d_{Lev}}$: The occurrences of ε in γ dictate which type of transition to take while the path ρ tells us which state to move to. As γ projected to the first component contains $L(s_2)\dots L(s_n)$ enriched with εs exactly $n-1$ transitions of type 1 or 3 are taken in τ obtained in this way and we indeed reach the last copy $\mathscr{T} \times \{n\}$. As ρ ends in a terminal state t_k, we furthermore reach the terminal state (t_k, n). Analogously, an infinite path ρ in $\mathscr{T}$ together with an edit sequence γ for $L(\pi)$ and $L(\rho)$ yields an infinite path in $\mathscr{T}^{\pi}_{d_{Lev}}$.

Based on these observations, we equip $\mathscr{T}^{\pi}_{d_{Lev}}$ with a weight function wgt on transitions: Transitions labeled with (σ,σ) for a $\sigma \in \Sigma$ get weight 0, the remaining transitions get weight 1.

Theorem 8. *Let $\mathscr{T} = (S, s_{init}, \rightarrow, L)$ be a transition system, E a set of terminal states, and C a set of states disjoint from E. Let $\Phi = \Diamond E$ or $\Phi = \Box\neg E$. Let $\pi = s_0 \dots s_n$ be an execution reaching C and satisfying Φ. It is decidable in polynomial time whether C is a d_{Lev}-counterfactual cause for Φ on π.*

Proof sketch. With the construction of the weighted transition system $\mathscr{T}^{\pi}_{d_{Lev}}$ above, the check can be done via the computation of shortest paths as for the Hamming distance above. □

3.3 Relation to Halpern-Pearl causality

In the sequel, we want to demonstrate how our definition of counterfactual causality relates to Halpern-Pearl-style definitions of causality in *structural equation models* [18, 19, 16]. A structural equation model consists of variables $X_1, \dots, X_n$ with finite domains that are governed by equations $X_i = f_i(X_1, \dots, X_{i-1}, C)$ for all $i \leq n$. Here, f_i is an arbitrary function for each i and C is an input parameter for the context. For our consideration, the context C does not play a role and we will hence omit it in the sequel. So, the value of variable X_i depends on the value of (some of) the variables with lower index and the dependency is captured by the function f_i. Halpern and Pearl use *interventions* to define causality for an effect E, which is a set of valuations of $X_1, \dots, X_n$. An intervention puts the value of a variable X_i to some α that is different from $f_i(X_1, \dots, X_{i-1})$, i.e., disregarding the equation f_i. Afterwards, the subsequent variables are evaluated as usual or by further interventions. Halpern and Pearl define:

Definition 9. Let $f_1, \dots, f_n$ over variables $X_1, \dots, X_n$ be a structural equation model as above and let E be an effect set of valuations such that the valuation of $X_1, \dots, X_n$ obtained by the structural equation model belongs to E. A *but-for-cause* is a minimal subset $X \subseteq \{X_1, \dots, X_n\}$ with the following property: There are values α_x for $x \in X$ such that putting variables $x \in X$ to α_x by intervention leads to a valuation of $X_1, \dots, X_n$ not exhibiting the effect E. More precisely, letting t_i be the valuation $[X_1 = w_1, \dots, X_{i-1} = w_{i-1}]$, where $w_i = f_i(w_1, \dots, w_{i-1})$ if $X_i \notin X$, and $w_i = \alpha_{X_i}$ if $X_i \in X$, we get that $t_{n+1} \notin E$.

In order to compare this to our notion of counterfactual causes, we view structural equation models as tree-like transition system $\mathcal{T}$: The nodes at level i are valuations for the variables $X_1, \ldots, X_{i-1}$. At each node s at level i, two actions are available: The action $\mathsf{default}$ moves to the state on level $i+1$ where the valuation in s is extended by setting X_i to the value $f_i(X_1, \ldots, X_{i-1})$ where the values for $X_1, \ldots, X_{i-1}$ are taken from the valuation in s. The action $\mathsf{intervention}$ extends the valuation of s by setting X_i to any other value than the action $\mathsf{default}$. The labelling in $\mathcal{T}$ assigns the label $\{\mathsf{intervention}\}$ to all states that are reached by the action $\mathsf{intervention}$. The remaining states and the initial state with the empty valuation get the label $\emptyset$. Given an effect E as a set of valuations, we interpret this as the corresponding set of leaf states in $\mathcal{T}$. The default path π that always chooses the action $\mathsf{default}$ corresponds to evaluating the equations in the structural equation model without interventions. We can now capture but-for-causality with d_{Hamm}-counterfactual causality along the default path π if all variables are Boolean:

Proposition 10. *Let $f_1, \ldots, f_n$ over Boolean variables $X_1, \ldots, X_n$ be a structural equation model and let E be an effect set of valuations. Let X be a but-for-cause for E. Let C_X be the set of all nodes in the transition system $\mathcal{T}$ which are reached by a* $\mathsf{default}$*-transition for a variable $x \in X$. Then, C_X is a d_{Hamm}-counterfactual cause for E in $\mathcal{T}$ on the default path π.*

For non-Boolean variables, the definitions of but-for-causes and of d_{Hamm}-counterfactual causes have one significant difference: A but-for-cause X merely requires the existence of values to assign to the variables in X by $\mathsf{intervention}$ such that the effect is avoided. A d_{Hamm}-counterfactual cause C in $\mathcal{T}$ requires that for all possible interventions on the variables in X, the effect is avoided. This universal quantification originates from the universal quantification over most similar worlds in the Stalnaker-Lewis semantics of counterfactual causality.

The minimality requirement of but-for-causes does not have a counterpart in the definition of d-counterfactual causes. This allows us to assert that a candidate set of states C is a d-counterfactual cause for an effect even if it contains redundancies. When trying to find d-counterfactual causes for a given effect, on the other hand, of course trying to find (cardinality-)minimal causes is a reasonable option.

Besides but-for causality, we can also capture actual causality as in [16] in our framework in the case of Boolean structural equation models. This is demonstrated in the extended version [31].

4 Counterfactual causality in reachability games

The counterfactual notion of causality introduced and investigated in the previous section can be applied to reachability games $\mathcal{G}$: We take the perspective of a player Π. Given a strategy σ for the opponent and a play in which Π lost, we apply the definition to the transition system obtained from σ and $\mathcal{G}$ and the given play. This allows us to analyze whether avoiding a certain set of states while playing against strategy σ as similarly as possible to the given play would have allowed Π to win. Depending on whether we take the perspective of Reach or Safe, the effect that the player loses the game is a safety or reachability property, which we considered as effects in transition systems. The need to be given a strategy for the opponent, however, constitutes a major restriction to the usefulness of this approach. All proofs omitted in this section can be found in the extended version [31].

4.1 D-counterfactual causality

We provide a definition of counterfactual causality in reachability games in the sequel in which we only need the strategy σ with which the player Π played and are interested in why the strategy σ allows the opponent to win the game. Since both players have optimal MD-strategies in a reachability game, we restrict ourselves to MD-strategies in the definition.

Definition 11. Let $\mathcal{G}$ be a reachability game with target set V_{Eff}. Let Π be one of the two players and let σ be a MD-strategy for player Π. Let C be a set of locations disjoint from V_{Eff}. Let D be a distance function on MD-strategies. We say that C is a D-counterfactual cause for the fact that Π loses using σ if

1. there are σ-plays that reach C on which Π loses,
2. there is an MD-strategy τ for player Π that avoids C (i.e., there is no τ-play reaching C),
3. all MD-strategies τ for player Π, that avoid C and that have minimal D-distance to σ among the strategies avoiding C, are winning for Π.

If we take the perspective of player Π in game $\mathcal{G}$ where the opponent $\bar{\Pi}$ does not control any locations, MD-strategies for Π satisfying condition 1 of the definition are essentially simple paths satisfying a safety or reachability effect property (with additional information on the states that are not visited by the path). To some extent, the definition can now be seen as a generalization of the definition for transition systems for suitable distance functions D: We say a strategy distance function D *generalizes a path distance function* d if in games where $\bar{\Pi}$ does not control any location, for all strategies σ,τ for Π, we have $D(\sigma,\tau) = d(\pi_\sigma,\pi_\tau)$ where π_σ and π_τ are the unique σ- and τ-plays. The definition that C is a D-counterfactual cause for σ losing the game agrees with the definition that C is a d-counterfactual cause on π_σ for $\lozenge V_{Eff}$ or $\Box\neg V_{Eff}$ in acyclic games in this case. In cyclic games, there is one caveat: The definition for games quantifies only over MD-strategies which induce a play that is a simple path or simple lasso. The definition for transition systems quantifies over more complicated paths as well.

Hausdorff distance d^H_{pref} based on the prefix metric d_{pref}. A way to obtain a strategy distance function generalizing a given path distance function is the use of the Hausdorff distance on the set of plays of the strategies [12, Section 6.2.2]: Let τ and σ be two MD-strategies, and d be a distance function over plays. The Hausdorff distance d^H based on d is defined by

$$d^H(\sigma,\tau) = \max\left\{ \sup_{\sigma\text{-plays } \pi} \inf_{\tau\text{-plays } \rho} d(\pi,\rho),\ \sup_{\tau\text{-plays } \rho} \inf_{\sigma\text{-plays } \pi} d(\pi,\rho) \right\}.$$

Let us consider the Hausdorff distance d^H_{pref} based on the prefix metric d_{pref} assuming that all states have a unique label. For two strategies σ and τ for Safe, the distance $d^H_{pref}(\sigma,\tau)$ is 2^{-n} where n is the least natural number such that there is a prefix of length n of a τ-play that is not a prefix of a σ-play, or vice versa. In order to find strategies that are as similar as possible to a given strategy σ, we hence have to consider strategies that follow σ for as many steps as possible. This leads to an algorithm for checking d^H_{pref}-counterfactual causality in reachability games that shares some similarities with the algorithm for checking d^{AP}_{pref}-counterfactual causality in transition systems.

Theorem 12. *Let $\mathcal{G} = (V, v_i, \Delta)$ where $V = V_{\mathsf{Reach}} \uplus V_{\mathsf{Safe}} \uplus V_{Eff}$ be a reachability game with target set V_{Eff} and σ a MD-strategy for player Π. Let C be a set of locations disjoint from V_{Eff}. We can check in polynomial time whether C is a d^H_{pref}-counterfactual cause for the fact that Π is losing using σ in $\mathcal{G}$.*

For the Hausdorff lifting of d_{Hamm} or d_{Lev}, the resulting notion of counterfactual causes in games is more complicated. If we try to adapt the approach used in transition systems, we need a way to capture the minimum distance of a given strategy to the closest winning strategies. However, shortest path games (as extension of the weighted transition systems used for d_{Hamm}- and d_{Lev}-counterfactual causes in transition systems) cannot be employed in an obvious way. In this paper, we now instead consider two further distance functions related to the Hamming distance for which we can provide algorithmic results.

Hamming strategy distance. Let σ and τ be two MD-strategies for Π in $\mathcal{G}$, we define the Hamming strategy distance by $d^s_{Hamm}(\sigma,\tau) = |\{v \in V \mid \sigma(v) \neq \tau(v)\}|$. As the Hamming distance on paths counts positions at which traces differ, the Hamming strategy distance counts positions at which two MD-strategies differ. Using a similar proof using shortest-path games [23] as for Theorem 5, we obtain the following polynomial-time result in the case of *aperiodic* games.

Theorem 13. *Let $\mathcal{G} = (V, v_i, \Delta)$ be an acyclic reachability game with target set V_{Eff} and σ a MD-strategy for player Π. Let C be a set of locations disjoint from V_{Eff}. We can check in polynomial time whether C is a d^s_{Hamm}-counterfactual cause for the fact that Π is losing using σ in $\mathcal{G}$.*

Hausdorff-inspired distance d^*. The distance function d^* computes the number of vertices where two MD-strategies make distinct choices along each play of both MD-strategies. It hence has some similarity to a Hausdorff-lifting of the Hamming distance on paths. This Hausdorff-lifting, however, counts the number of *occurrences* of vertices at which two paths differ (in their label). Instead, for a play $\rho = v_0 v_1 \dots$ and a strategy σ for Π, we define the *distance between σ and ρ $dist(\rho,\sigma)$* as the number of vertices $v \in V_\Pi$ (i.e., not the number of occurrences) such that there exists $i \in \mathbb{N}$ with $v = v_i$ in ρ, and $\sigma(v_i) \neq (v_i, v_{i+1})$.We define d^* for two strategies τ, σ by

$$d^*(\tau,\sigma) = \max\big(\sup_{\rho\mid\tau\text{-play}} dist(\rho,\sigma),\ \sup_{\pi\mid\sigma\text{-play}} dist(\pi,\tau)\big).$$

To simplify the notation, we define $d^\tau(\sigma) = \sup_{\rho\mid\tau\text{-play}} dist(\rho,\sigma)$. We prove that the threshold problem for d^* is NP-complete via a reduction from the *longest simple path problem*:

Proposition 14. *Let $\mathcal{G}$ be a reachability game, σ, τ be two MD-strategies for Π, and $k \in \mathbb{N}$ be a threshold. Then deciding if $d^*(\tau,\sigma) \geq k$ is NP-complete.*

The proposition explains why understanding d^*-counterfactual causes is complex. We leave a further investigation of such notions for future work. As a first step toward a better understanding, we turn our attention to a conceptually simpler notion, the explanation induced by a counterfactual cause.

Example 15. Let us illustrate counterfactual causes according to distances on strategies. We consider the reachability game depicted in the left of Figure 2 and the non-winning strategy σ for Reach depicted in green. Under d^H_{pref} or d^*, the counterfactual cause for Reach is $\{v_2, v_3\}$. Indeed, there exists one play that reaches v_3 and loses for Reach, and there exists a unique strategy that avoids $\{v_2, v_3\}$ by changing the choice of σ in v_1. Moreover, this counterfactual cause is minimal since $\{v_3\}$ is not a cause. Indeed, the (losing) strategy that differs from σ in v_0 and v_1 avoids $\{v_3\}$ with a minimal distance to σ, i.e. 2^{-2} for d^H_{pref} and 1 for d^*. Under d^s_{Hamm}, the counterfactual cause for Reach is $\{v_3\}$. Indeed, two strategies exist with a distance of 1 to σ according to the vertex where Reach changes its decision. In these two strategies, only one avoids $\{v_3\}$: the strategy where Reach change its decision in v_1. ⌟

4.2 D-counterfactual explanation

Given a D-counterfactual cause, we want to explain what is wrong in the losing strategy for Π. In particular, we are interested in sets of locations C such that Π could have won the game if she had not made the decisions of σ in the locations in C.

Definition 16. Let $\mathcal{G}$ be a reachability game and σ be a non-winning MD-strategy for Π. Let $E \subseteq V_\Pi$. We call E an *explanation* in $\mathcal{G}$ under σ if there exists a winning MD-strategy τ such that for all vertices $v \in V_\Pi$, $\tau(v) = \sigma(v)$ iff $v \notin E$. We call such a τ an *E-distinct σ-strategy*.

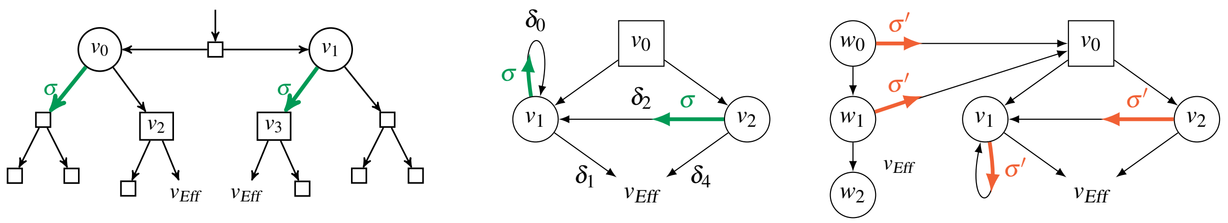


Figure 2: On the left and in the middle, two reachability games with initial vertex v_0 and strategy σ for Reach (depicted in green). On the right, the reachability game obtained by reduction of Corollary 21 from the game depicted in the middle with initial vertex w_0 and σ' be a non winning strategy for Reach.

We note that the definition of an explanation does not refer to a distance function. However, given a D-counterfactual cause, we can compute an explanation no matter which distance D is used.

Proposition 17. *Let $\mathscr{G} = (V, v_i, \Delta)$ be a reachability game, D a distance function on strategies and σ be a non-winning MD-strategy for Π. Let $C \subseteq V$ be a D-counterfactual cause. We can compute an explanation E (from C) in polynomial time.*

Proof. Let $\mathscr{G}' = (V \setminus C, v_i, \Delta)$ be the reachability game. Since D is a D-counterfactual cause, we know that there exists a winning strategy τ in $\mathscr{G}'$. We can compute this strategy in time polynomial in the size of $\mathscr{G}'$ with the attractor method and we define $E = \{v \mid \sigma(v) \neq \tau(v)\}$. □

A winning strategy differing from σ in E might not have much in common with σ. For this reason, explanations that point out changes in the decisions of σ in E that enforce only the minimal necessary change to obtain a winning strategy τ from σ are of particular interest. We can use a distance function D to quantify how much a strategy needs to be changed.

Definition 18. Let $\mathscr{G}$ be a reachability game and σ be a non-winning MD-strategy for Π. For a distance function D for MD-strategies, we call a explanation E a *D-minimal explanation*, if there exists a winning E-distinct σ-strategy τ with $d(\tau, \sigma) = \min\{d(\mu, \sigma) \mid \mu \text{ is a winning MD-strategy for } \Pi\}$.

For a strategy σ and an explanation E, the distance $d^s_{Hamm}(\sigma, \tau)$ for an E-distinct σ-strategy τ is precisely $|E|$. So, d^s_{Hamm}-minimal explanations are cardinality-minimal explanations.

Example 19. Let us illustrate explanations and D-minimal explanations. We consider the reachability game $\mathscr{G}$ where Reach wins depicted in the left of Figure 2 with σ, a non-winning MD-strategy for Reach, depicted in green. We note that $E = \{v_1, v_2\}$ is an explanation in $\mathscr{G}$ under σ. A winning E-distinct σ-strategy τ for Reach is given by $\tau(v_1) = \delta_1$ and $\tau(v_2) = \delta_4$. However, E is not a d^*-minimal explanation or d^s_{Hamm}-minimal explanation. Clearly, $d^s_{Hamm}(\tau, \sigma) = 2$. Further, also $d^*(\tau, \sigma) = 2$ as the σ-play $v_0 v_2 v_1^\omega$ visits two states, namely v_2 and v_1 at which σ and τ make different decisions. The set $E' = \{v_1\}$, however is a d^*-minimal explanation and d^s_{Hamm}-minimal explanation: the E'-distinct σ-strategy τ' choosing δ_1 in v_1 and behaving like σ in v_2 wins and has d^s_{Hamm}- and d^*-distance 1 to σ. As any winning strategy has at least distance 1 to σ, E' is hence a D-minimal explanation for both distance functions. ⌟

For D-minimal explanations, it is central to find a winning MD-strategy that minimises the distance D to the given losing strategy σ. We take a look at this problem from the point of view of Reach and prove that for d^s_{Hamm} and d^* the associated threshold problems are not in P if P$\neq$NP.

Theorem 20. *Given a game $\mathscr{G}$, a losing strategy σ for* Reach*, and $k \in \mathbb{N}$, deciding if there exists a winning MD-strategy τ for* Reach *such that $d^s_{Hamm}(\tau, \sigma) \leq k$ is NP-complete. Further, the problem whether there is a winning MD-strategy τ with $d^*(\tau, \sigma) \leq k$ is not in P if P$\neq$NP.*

Proof sketch. To establish the NP upper bound for d^s_{Hamm}, we can guess a MD-strategy τ for Reach and check in polynomial time whether it is winning and whether $d^s_{Hamm}(\tau,\sigma) \leq k$. For the NP-hardness for d^s_{Hamm}, we provide a polynomial-time many-one reduction from the NP-complete decision version of the *feedback vertex set* [22]. Given a cyclic (directed) graph G, this problem asks whether there is a set S of size at most k such that if we remove this set, $G \setminus S$ becomes acyclic. For the problem for d^*, we provide a polynomial-time Turing reduction from the same problem. A detailed proof is given in [31]. □

We deduce that checking D-minimality of an explanation cannot be done in polynomial time if P≠NP.

Corollary 21. *Let $\mathcal{G}$ be a reachability game, σ be a non-winning MD-strategy for* Reach*, and $E \subseteq V$. The problem to check if E is a d^s_{Hamm}-minimal explanation in $\mathcal{G}$ for σ is coNP-complete. The problem to check if E is a d^*-minimal explanation in $\mathcal{G}$ for σ is not in P if P≠NP.*

Despite the hardness in the general case, if $\mathcal{G}^\sigma$ is acyclic, we prove that we can compute the winning MD-strategy that minimises the d^*-distance to σ in polynomial time. From this strategy, a d^*-minimal explanation can then be computed as in Proposition 17. The proof of Theorem 22 (in the extended version [31]) constructs a shortest-path game [23] without negative weights in which an optimal strategy, that leads to the desired winning strategy in the original game, can be computed in polynomial time.

Theorem 22. *Let $\mathcal{G}$ be reachability game where* Reach *wins, and σ be a non-winning MD-strategy for* Reach *such that $\mathcal{G}^\sigma$ is acyclic. Then, we can compute a winning MD-strategy τ that minimizes the distance d^* to σ in polynomial time.*

5 Conclusion and Outlook

The notion of d-counterfactual cause for a distance function d in transition systems turned out to be checkable in polynomial time for the distance functions d_{pref}, d_{Hamm}, and d_{Lev} and so it has the potential to be employed in efficient tools to provide understandable explanations of the behavior of a system. In our algorithmic results for safety effects Φ, one caveat remains: we only considered finite executions reaching a cause candidate C and satisfying Φ. Allowing also finitely representable, e.g., ultimately periodic paths, constitutes a natural extension, which requires adjustments in the provided algorithms.

The problem of finding good causes remains as future work: Whenever causality can be checked in polynomial time, there is an obvious non-deterministic polynomial-time upper bound on the problem to decide whether there are causes below a given size, but the precise complexities are unclear. A further idea is to use the distance function to assess how good a cause is by considering the distance from the actual execution to the closest executions avoiding a cause. For reachability effects and the prefix and Hamming distance, the set of direct predecessors optimizes this distance. For other distance functions or safety causes, this measure could, nevertheless, be more useful. The search for similar measures for the quality of causes constitutes an interesting direction for future work.

In reachability games, we saw that the analogous definition of D-counterfactual causes can be checked in polynomial time for the Hausdorff-lifting d^H_{pref} of the prefix metric, as well. For other distance functions, the definition seems to lead to complicated notions due to the involved quantification over all MD-strategies avoiding the cause and having a minimal distance to a given strategy. A closer investigation of these notions might, nevertheless, be a fruitful subject for future research. However, our analysis of the conceptually simpler D-minimal explanations provides insights into the complications one might encounter here. For the Hausdorff-inspired distance function d^*, we showed that already the threshold problem for the distance between two given MD-strategies is NP-hard. Furthermore, for the relatively simple distance function d^s_{Hamm}, checking the d^s_{Hamm}-minimality of an explanation is in coNP-complete. For the Hausdorff-inspired distance function d^*, checking d^*-minimality is not in P unless P=NP.

References

[1] Christel Baier, Norine Coenen, Bernd Finkbeiner, Florian Funke, Simon Jantsch & Julian Siber (2021): *Causality-based game solving*. In: *International Conference on Computer Aided Verification*, Springer, pp. 894–917, doi:10.1007/978-3-030-81685-8_42.

[2] Christel Baier, Clemens Dubslaff, Florian Funke, Simon Jantsch, Rupak Majumdar, Jakob Piribauer & Robin Ziemek (2021): *From Verification to Causality-Based Explications (Invited Talk)*. In Nikhil Bansal, Emanuela Merelli & James Worrell, editors: *48th International Colloquium on Automata, Languages, and Programming, (ICALP)*, *LIPIcs* 198, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 1:1–1:20. Available at `https://doi.org/10.4230/LIPIcs.ICALP.2021.1`.

[3] Christel Baier, Clemens Dubslaff, Florian Funke, Simon Jantsch, Jakob Piribauer & Robin Ziemek (2022): *Operational Causality – Necessarily Sufficient and Sufficiently Necessary*. In Nils Jansen, Mariëlle Stoelinga & Petra van den Bos, editors: *A Journey from Process Algebra via Timed Automata to Model Learning : Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday*, Springer Nature Switzerland, Cham, pp. 27–45, doi:10.1007/978-3-031-15629-8_2.

[4] Christel Baier, Florian Funke & Rupak Majumdar (2021): *A Game-Theoretic Account of Responsibility Allocation*. In Zhi-Hua Zhou, editor: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, International Joint Conferences on Artificial Intelligence Organization, pp. 1773–1779, doi:10.24963/ijcai.2021/244. Main Track.

[5] Christel Baier & Joost-Pieter Katoen (2008): *Principles of Model Checking*. MIT Press.

[6] Thomas Ball, Mayur Naik & Sriram K. Rajamani (2003): *From Symptom to Cause: Localizing Errors in Counterexample Traces*. *SIGPLAN Not.* 38(1), pp. 97–105, doi:10.1145/640128.604140.

[7] Ilan Beer, Shoham Ben-David, Hana Chockler, Avigail Orni & Richard J. Trefler (2012): *Explaining counterexamples using causality*. *Formal Methods in System Design* 40(1), pp. 20–40, doi:10.1007/s10703-011-0132-2.

[8] Hana Chockler (2016): *Causality and Responsibility for Formal Verification and Beyond*. In Gregor Gößler & Oleg Sokolsky, editors: *Proceedings First Workshop on Causal Reasoning for Embedded and safety-critical Systems Technologies, CREST@ETAPS 2016, Eindhoven, The Netherlands, 8th April 2016*, *EPTCS* 224, pp. 1–8, doi:10.4204/EPTCS.224.1.

[9] E. M. Clarke, O. Grumberg & D. Peled (1999): *Model Checking*. MIT Press.

[10] Edmund M. Clarke, Orna Grumberg, Kenneth L. McMillan & Xudong Zhao (1995): *Efficient Generation of Counterexamples and Witnesses in Symbolic Model Checking*. In: *Proc. of the 32nd Annual ACM/IEEE Design Automation Conf. (DAC)*, ACM, New York, NY, USA, pp. 427–432, doi:10.1145/217474.217565.

[11] Norine Coenen, Bernd Finkbeiner, Hadar Frenkel, Christopher Hahn, Niklas Metzger & Julian Siber (2022): *Temporal Causality in Reactive Systems*. In: *20th International Symposium on Automated Technology for Verification and Analysis, ATVA*, pp. 25–28, doi:10.1007/978-3-031-19992-9_13.

[12] M.C. Delfour & J.P. Zolesio (2011): *Shapes and Geometries: Metrics, Analysis, Differential Calculus, and Optimization, Second Edition*. Advances in Design and Control, Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), doi:10.1137/1.9780898719826. Available at `https://books.google.fr/books?id=fjjvX9a9cxUC`.

[13] Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors (2002): *Automata Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag, Berlin, Heidelberg.

[14] Alex Groce, Sagar Chaki, Daniel Kroening & Ofer Strichman (2006): *Error explanation with distance metrics*. *International Journal on Software Tools for Technology Transfer* 8(3), pp. 229–247, doi:10.1007/978-3-540-24730-2_8.

[15] Alex Groce & Willem Visser (2003): *What Went Wrong: Explaining Counterexamples*. In Thomas Ball & Sriram K. Rajamani, editors: *Model Checking Software*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 121–136, doi:10.1007/3-540-44829-2_8.

[16] Joseph Y. Halpern (2015): *A Modification of the Halpern-Pearl Definition of Causality*. In: Proc. of the 24th Intern. Joint Conf. on AI (IJCAI), AAAI Press, pp. 3022–3033.

[17] Joseph Y. Halpern & Judea Pearl (2001): *Causes and Explanations: A Structural-Model Approach: Part i: Causes*. In: Proc. of the 17th Conf. on Uncertainty in AI (UAI), Morgan Kaufmann Publishers Inc., pp. 194–202, doi:10.1093/bjps/axi147.

[18] Joseph Y. Halpern & Judea Pearl (2005): *Causes and Explanations: A Structural-Model Approach. Part I: Causes*. The British Journal for the Philosophy of Science 56(4), pp. 843–887, doi:10.1093/bjps/axi147.

[19] Joseph Y. Halpern & Judea Pearl (2005): *Causes and Explanations: A Structural-Model Approach. Part II: Explanations*. The British Journal for the Philosophy of Science 56(4), pp. 889–911, doi:10.1093/bjps/axi148.

[20] David Hume (1739): *A Treatise of Human Nature*. John Noon, doi:10.1093/oseo/instance.00032872.

[21] David Hume (1748): *An Enquiry Concerning Human Understanding*. London.

[22] Richard M. Karp (1972): *Reducibility among Combinatorial Problems*, pp. 85–103. Springer US, Boston, MA, doi:10.1007/978-1-4684-2001-2_9.

[23] Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled Elbassioni, Vladimir Gurvich, Gabor Rudolf & Jihui Zhao (2007): *On Short Paths Interdiction Problems: Total and Node-Wise Limited Interdiction*. Theory of Computing Systems, doi:10.1007/s00224-007-9090-x.

[24] Florian Leitner-Fischer & Stefan Leue (2013): *Causality Checking for Complex System Models*. In: Proc. of the 14th Intern. Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI), pp. 248–267, doi:10.1007/978-3-642-35873-9_16.

[25] Vladimir I. Levenshtein (1966): *Binary codes capable of correcting deletions, insertions, and reversals*. Soviet physics doklady 10(8), pp. 707–710.

[26] David Lewis (1973): *Causation*. Journal of Philosophy 70(17), pp. 556–567, doi:10.2307/2025310.

[27] David K. Lewis (1973): *Counterfactuals*. Cambridge, MA, USA: Blackwell.

[28] J. L. Mackie (1965): *Causes and Conditions*. American Philosophical Quarterly 2(4), pp. 245–264. Available at `http://www.jstor.org/stable/20009173`.

[29] Z. Manna & A. Pnueli (1995): *The Temporal Logic of Reactive and Concurrent Systems: Safety*. Springer-Verlag.

[30] Kedar S. Namjoshi (2001): *Certifying Model Checkers*. In: 13th International Conference on Computer Aided Verification (CAV), Lecture Notes in Computer Science 2102, Springer, pp. 2–13. Available at `https://doi.org/10.1007/3-540-44585-4_2`.

[31] Julie Parreaux, Jakob Piribauer & Christel Baier (2023): *Counterfactual Causality for Reachability and Safety based on Distance Functions*. arXiv:2308.11385. ArXiv preprint: `arxiv.org/abs/2308.11385`.

[32] Judea Pearl (2009): *Causality*, 2 edition. Cambridge University Press, doi:10.1017/CBO9780511803161.

[33] Jonas Peters, Dominik Janzing & Bernhard Schölkopf (2017): *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press, Cambridge, MA, USA.

[34] Ibo van de Poel (2011): *The Relation Between Forward-Looking and Backward-Looking Responsibility*, pp. 37–52. Springer Netherlands, Dordrecht, doi:10.1007/978-94-007-1878-4_3.

[35] Manos Renieres & Steven P. Reiss (2003): *Fault localization with nearest neighbor queries*. In: Proc. of the 18th IEEE Intern. Conf. on Automated Software Engineering (ASE), pp. 30–39, doi:10.1109/ASE.2003.1240292.

[36] Klaus U Schulz & Stoyan Mihov (2002): *Fast string correction with Levenshtein automata*. International Journal on Document Analysis and Recognition 5(1), pp. 67–85, doi:10.1007/s10032-002-0082-8.

[37] Robert C. Stalnaker (1968): *A Theory of Conditionals*. In William L. Harper, Robert Stalnaker & Glenn Pearce, editors: IFS. The University of Western Ontario Series in Philosophy of Science, 15, Springer, Dordrecht, pp. 41–55, doi:10.1007/978-94-009-9117-0_2.

[38] Chao Wang, Zijiang Yang, Franjo Ivancic & Aarti Gupta (2006): *Whodunit? Causal Analysis for Counterexamples*. In: *Proc. of the 4th Intern. Symp. on Automated Technology for Verification and Analysis (ATVA)*, pp. 82–95, doi:10.1007/11901914_9.

[39] Shaohui Wang, Anaheed Ayoub, BaekGyu Kim, Gregor Gößler, Oleg Sokolsky & Insup Lee (2013): *A Causality Analysis Framework for Component-Based Real-Time Systems*. In: *Proceedings of the 4th International Conference on Runtime Verification (RV)*, pp. 285–303, doi:10.1007/978-3-642-40787-1_17.

[40] Vahid Yazdanpanah & Mehdi Dastani (2016): *Distant Group Responsibility in Multi-agent Systems*. In Matteo Baldoni, Amit K. Chopra, Tran Cao Son, Katsutoshi Hirayama & Paolo Torroni, editors: *PRIMA 2016: Princiles and Practice of Multi-Agent Systems - 19th International Conference, Phuket, Thailand, August 22-26, 2016, Proceedings*, *Lecture Notes in Computer Science* 9862, Springer, pp. 261–278, doi:10.1007/978-3-319-44832-9_16.

[41] Vahid Yazdanpanah, Mehdi Dastani, Wojciech Jamroga, Natasha Alechina & Brian Logan (2019): *Strategic Responsibility Under Imperfect Information*. In Edith Elkind, Manuela Veloso, Noa Agmon & Matthew E. Taylor, editors: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 592–600. Available at `http://dl.acm.org/citation.cfm?id=3331745`.

[42] Andreas Zeller (2002): *Isolating Cause-Effect Chains from Computer Programs*. In: *Proc. of the 10th ACM SIGSOFT Symp. on Foundations of Software Engineering (FSE)*, ACM, New York, NY, USA, pp. 1–10, doi:10.1145/587051.587053.

Conflict-Aware Active Automata Learning

Tiago Ferreira   Léo Henry   Raquel Fernandes da Silva
University College London
London, UK
{t.ferreira,leo.henry,raquel.silva.20}@ucl.ac.uk

Alexandra Silva
Cornell University
Ithaca, NY, USA
alexandra.silva@cornell.edu

Active automata learning algorithms cannot easily handle *conflict* in the observation data (different outputs observed for the same inputs). This inherent inability to recover after a conflict impairs their effective applicability in scenarios where noise is present or the system under learning is mutating.

We propose the Conflict-Aware Active Automata Learning (C$\mathcal{E}$AL) framework to enable handling conflicting information during the learning process. The core idea is to consider the so-called observation tree as a first-class citizen in the learning process. Though this idea is explored in recent work, we take it to its full effect by enabling its use with any existing learner and minimizing the number of tests performed on the system under learning, specially in the face of conflicts. We evaluate C$\mathcal{E}$AL in a large set of benchmarks, covering over 30 different realistic targets, and over 18,000 different scenarios. The results of the evaluation show that C$\mathcal{E}$AL is a suitable alternative framework for closed-box learning that can better handle noise and mutations.

1 Introduction

Formal methods have a long history of success in the analysis of critical systems through abstract models. These methods are rapidly expanding their range of applications and recent years saw an increase in industrial teams applying them to (large-scale) software [6, 9, 10, 12, 13, 25]. The applicability of such methods is limited by the availability of good models, which require time and expert knowledge to be hand-crafted and updated. To overcome this issue, a research area on automatic inference of models, called *model learning* [32], has gained popularity. Broadly, there are two classes of model learning: *passive learning*, which attempts to infer a formal model from a static log, and *active learning*, where interaction with the system is allowed to refine knowledge during the inference.

In this paper, we focus on active learning, motivated by its successful use in verification tasks, e.g. in analyzing network protocol implementations, as TCP [18], SSH [19], and QUIC [15], or understanding the timing behavior of modern CPUs [34]. Current state-of-the-art active learning algorithms rely on the *Minimally Adequate Teacher* (MAT) framework [4], which formalizes a process with two agents: a *learner* and a *teacher*. The learner tries to infer a formal model of a system, and the teacher is omniscient of the system, being able to answer queries on potential behaviors and the correctness of the learned model. MAT assumes that the interactions between both agents are perfect and deterministic.

Learning In Practice Interactions with the *System Under Learning* (SUL) are often non-deterministic in some way, e.g. the communications can be noisy (i.e. query answers do not only reflect the actual system output, but are instead a consequence of its interaction with the environment), or the SUL itself can change during learning. This can lead to *conflicts*, which we define in the following way:

A *conflict* appears when a query's answer formally contradicts a previous query in a way that cannot be expressed by a model of the target class.

Current MAT Learners cannot handle the conflicts that arise during learning. Thus, when used in practice, MAT learner implementations use artifacts to circumvent conflicting observations.

For example, in the case of noise, each interaction has a chance of diverging from its usual behavior. To handle this, MAT learners repeat each query n times and majority-vote the result. They aim to guess an n sufficiently large to prevent *any* noisy observation from reaching the learner, but small enough to let the computation finish before timeout. As a consequence, noise threatens both *efficiency* and *correctness* of learning. We provide a framework alleviating this issue without tailoring it to specific MAT learners.

Irrespective of the nature of the conflicts detected, dealing with them requires the ability to *backtrack* certain decisions that were made based on what is now considered incorrect information. This pinpoints the issue with current MAT learners: there is no notion of information storage other than the internal data structure that the learners use to build the model, which is not easily updatable in the face of conflict. This structure in fact needs to be fully rebuilt if a conflict is found, generating many superfluous (and expensive!) queries to the SUL. Separating the learning process from the information gathered through the queries allows us to *retain* all the previous non-conflicting information. This alleviates the main cost of conflict handling: the unnecessary repetition of tests on the system. The Learner then only needs to rebuild its data structure based on the information already available.

Contribution Based on the ideas above, this paper proposes the *Conflict Aware Active Automata Learning* (C$\mathcal{E}$AL, pronounced *seal*) framework. Any existing MAT learner can be used in C$\mathcal{E}$AL. When a conflict arises, we provide a method for updating the learner's internal state — without making assumptions on its data-structure — so that it remains conflict-free while removing only inconsistent information.

In a nutshell, this paper aims to provide classic MAT learners with a way to recover from conflicts caused by either noise or potential mutations of the system.

At the heart of C$\mathcal{E}$AL is the use of an *observation tree*, a data structure (external to the learner) used to store information gathered from the SUL. It can be efficiently updated and used by the learner to construct its own internal data structure. When a conflict appears, we update the observation tree to reflect our knowledge, while the learner's data structure is *pruned* to a conflict-free point and then expanded from the observation tree. Crucially, the learner uses *the observations already stored in the tree* without requiring tests on the SUL for already observed behaviors. C$\mathcal{E}$AL's main features are:

- The SUL is a first-class citizen, instead of being abstracted. C$\mathcal{E}$AL notably does not rely on *equivalence queries*, replacing them with either a check of the stored knowledge (when sufficient) or an *equivalence test*, using an m-complete testing algorithm (e.g. the Wp-method [20] or Hybrid-ADS [26,29]).
- The information obtained through tests on the SUL is stored in an observation tree managed by a new *Reviser* agent that is responsible for handling the conflicts and answering the learner's queries like a teacher. Providing a teacher interface is an important aspect as it enables the use of any MAT-based algorithm seamlessly, only requiring the ability to restart a classic MAT learner.
- The Reviser alone interacts with the SUL by means of tests meant to expand its observation tree.

Crucially, C$\mathcal{E}$AL is less abstract than MAT, representing directly the objects and challenges of *practical* active learning, while still allowing the design of Learners to enjoy the simplifying abstraction of MAT.

After some preliminaries in Section 2 we formalize and prove the above claims in Section 3. We evaluate C$\mathcal{E}$AL in Section 4 using a broad range of experiments [27]. We compare several state-of-the-art algorithms (namely `L`* [4], `KV` [24], `TTT` [22] and `L`$^\#$ [33]) for targets of different sizes and different levels of noise, while varying the controllable parameters for both MAT and C$\mathcal{E}$AL. The experimental

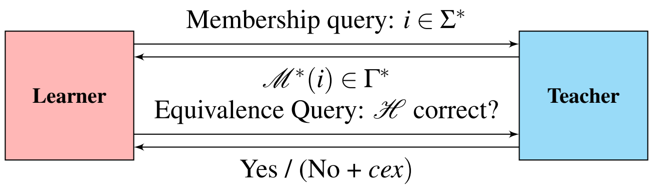

Membership query: $i \in \Sigma^*$
Learner
$\mathscr{M}^*(i) \in \Gamma^*$
Equivalence Query: $\mathscr{H}$ correct?
Teacher
Yes / (No + *cex*)

Figure 1: The Minimally Adequate Teacher framework.

results show that in the case of noise, C$\mathcal{E}$AL allows us to drastically reduce the number of repeats required to learn correct models by handling some conflicts in the information it gathers from the system. This allows C$\mathcal{E}$AL to achieve a success rate of 95.5% compared to MAT's 79.5% in our experiments.

A long version of this paper, complete with the appendices presenting the proofs, some more formalization, extensive experimental results and some more discussion can be found in [16]. References are given when it can be useful.

2 Preliminaries

In this section, we recall Mealy machines and MAT. Fix an alphabet A (a finite set of symbols). The set of finite words is denoted A^*, the empty word ε, and the set of non-empty words by A^+. The length of a word $w \in A^*$ is denoted $|w|$, its sets of prefixes by $\mathsf{prefixes}(w)$, its k-th element by $w[k]$ and the subword from the i-th to the j-th element by $w[i,j]$. The concatenation of word w with symbol a is denoted by wa.

Mealy Machines For the rest of the paper, we fix an input and output alphabet pair (Σ,Γ). A *Mealy machine* over alphabets (Σ,Γ) is a tuple $\mathscr{M} = (Q, q_0, \delta, \lambda)$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is a transition function and $\lambda : Q \times \Sigma \to \Gamma$ an output function. Mealy machines assign *output words* ($o \in \Gamma^*$) to *input words* ($i \in \Sigma^*$)—one reads input letters using δ and collects all output letters given by λ. This is achieved using inductive extensions of δ and λ:

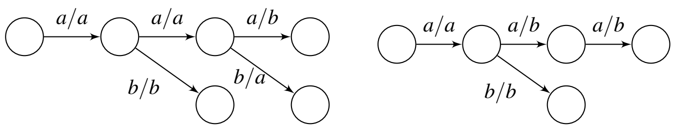
$$\delta^* : Q \times \Sigma^* \to Q \qquad \delta^*(q,\varepsilon) = q \qquad \delta^*(q, ia) = \delta(\delta^*(q,i), a)$$
$$\lambda^+ : Q \times \Sigma^+ \to \Gamma \qquad \lambda^+(q, ia) = \lambda(\delta^*(q,i), a)$$

We now build the semantics function $\mathscr{M}^* : \Sigma^* \to \Gamma^*$ given by

$$\mathscr{M}^*(a_1 \cdots a_j) = b_1 \cdots b_j \quad \text{where } b_k = \lambda^+(q_0, a_1 \cdots a_k), \text{ for all } k = 1, \dots, j.$$

Note the preservation of length of input words in output. When the functions δ and λ are partial we call the Mealy machine $\mathscr{M}$ partial. A partial tree-shaped Mealy machine is called an *observation tree*.

Example 1. *On the left below is the tree representing the tests $\{(aaa, aab), (aab, aaa), (ab, ab)\}$ and on the right the tree representing $\{(aaa, abb), (ab, ab)\}$.*



Active Model Learning Active learning is a process in which a *learner* can interact with an omniscient *teacher* to build a model of an unknown system. Formally, this type of learning uses the *Minimally Adequate Teacher* (MAT) framework [4] (see Fig. 1). The teacher is supposed to have enough knowledge about the target machine $\mathscr{M}$ to be able to answer two types of queries:

Membership The learner sends an input word i to the teacher, who answers with the output word $\mathscr{M}^*(i)$.

Equivalence The learner proposes a hypothesis model $\mathscr{H}$. The teacher either confirms the model as correct or provides a counterexample $cex \in \Sigma^*$ such that $\mathscr{H}^*(cex) \neq \mathscr{M}^*(cex)$.

The MAT framework is an interesting abstraction to design algorithms and conduct proofs, and has been the basis for active model learning since its introduction (see e.g. `L*` [4], `KV` [24], `TTT` [22] or `L#` [33]). The teacher abstracts the system under learning (SUL), which complicates discussions on the practical interfaces between the learner and the SUL during applications. MAT does not separate the learner's core features (i.e. choosing the queries to be made and building hypotheses) from the storage of observations. This has led the community to resort to *caches*, often implemented through observation trees, to access observations directly. Being mostly tricks to avoid repeating queries, caches are rarely discussed in the literature (although used during experiments), which had so far delayed a discussion on the practical implications of a proper handling of observations. This paper addresses this.

Noise on communications The term *noise* is usually used to described a wide range (if not any form) of perturbations that can happen between the designed agent (in our case the Learner) and the SUL. In the case of this study we are primarily interested in the classification between *input* and *output* noise.

Output noise We call output noise a perturbation that only affects what our agent sees from the world, i.e. the outputs of the SUL. Formally, this kind of noise can be represented as a non-deterministic function of $\mathscr{M}^*(i)$ returning a different output word of same size.

Input noise This kind of noise instead affects the query i inputted into the SUL, so that a different input word i' of same size is processed instead.

Noise can have different levels of *structure*, being generated by different kinds of models or probability distributions. As this paper strives for a generic approach, no assumption is made on the structure of noise. Furthermore, experiments will use generic noise that has a fixed probability *for each symbol* of the word, taken in sequence, to replace it with a random one according to a uniform distribution. One notable restriction of our approach is that it does not target adversarial modifications — such as an attacker trying to change the Learner's hypotheses.

Remark 1. *We do not further formalize noise, as it stems for very practical considerations that may require a wide array of different formalizations. The method we propose is* generic *and aims to demonstrate that paying attention to noise and* conflicts *allows significant efficiency gains without any specialization towards a specific model of noise.*

3 Conflict Aware Active Automata Learning

We now introduce our alternative to MAT in practice — the *Conflict-Aware Active Automata Learning* ($C\mathcal{E}AL$) framework. $C\mathcal{E}AL$'s main features are as follows:

- The SUL is a first class citizen, allowing for clearer practical discussions and modularity.
- The information obtained through tests on the SUL is stored in a new *Reviser* agent that handles the conflicts and answers the learner's queries like a teacher.
- The Reviser alone interacts with the SUL by means of tests meant to expand its observation tree. The learner's queries are answered from the observation tree.

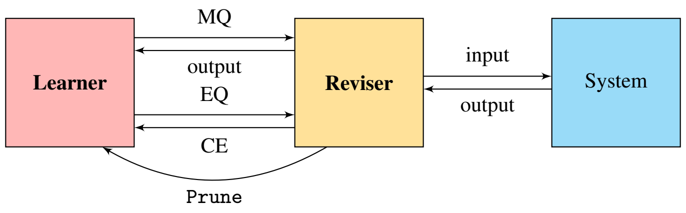


Figure 2: Simplified view of the C$\mathcal{E}$AL framework. See Fig. 4 and 5 for more detail.

3.1 Framework Overview

C$\mathcal{E}$AL (Fig. 2) is centered around three agents — the Learner, the System (SUL), and the Reviser — and the interfaces between them. The **Learner** plays the same general role as in MAT. Crucially, any MAT learner can be used in C$\mathcal{E}$AL (e.g. `L`*, `KV`, `TTT`, `L`$^\#$). The Learner *does not have* to store the information obtained from tests on the system. It focuses on the questions "What is the next query to make?" and "How is the hypothesis built?". The **System** is the System Under Learning, together with its environment (e.g. noise). The **Reviser** handles knowledge and conflicts. It answers the question "What do we know about the system?". It is set between the Learner and the System with interfaces to both of them.

C$\mathcal{E}$AL is designed to improve the practical learning of reactive systems like Mealy machines. As such, it makes use of features that are core to such models, like causality and closure under inputs and outputs. However, the main ideas behind C$\mathcal{E}$AL's philosophy and separation of concerns can be adapted to learn other types of automata, such as acceptors like DFAs.

Remark 2. *The Reviser acts as a MAT teacher w.r.t. the Learner, answering membership and equivalence queries, with the added ability to* `Prune` *the learner to place it in a state coherent with the Reviser's information. On the System's view, the Reviser acts as a tester, providing input sequences (tests) and recording the system output. The outside views of the learner and system in* C$\mathcal{E}$AL *are illustrated below.*

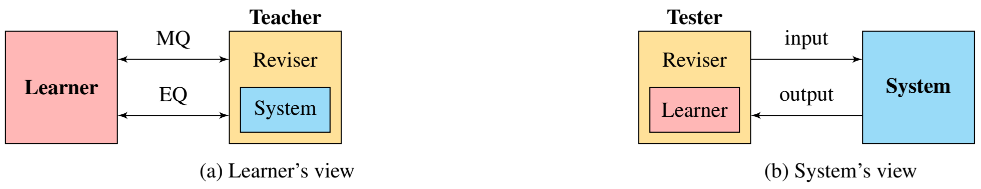


Figure 3: Reviser's interfaces

The **interfaces** on the Learner side are similar to MAT: the Learner can perform membership queries (MQ) and equivalence queries (EQ) on the Reviser, with the latter potentially resulting in a counterexample (CE). Note that the queries are sent to the Reviser and not directly to the SUL: a crucial design choice. This allows us to control the information that the Learner obtains, and reuse the information in the Reviser with no new tests. Formally, C$\mathcal{E}$AL provides the following functions as module interfaces:

- `MQ` $: \Sigma^* \to (\Gamma^* \cup \{$`Prune`$\})$ the membership query of the learner that the Reviser has to implement. It varies from the MAT function as the Reviser may return a command to prune the learner's state instead of an output.
- `EQ` $: \mathsf{Mealy} \to ((\Sigma^* \times \Gamma^*) \cup \{$`Prune`$\})$ the equivalence query of the learner that the Reviser has to implement. It may return "`Prune` " instead of "Yes".
- `System` $: \Sigma^* \to (\Sigma^* \times \Gamma^*)$ is a call to the system for a specific test. The system returns the corresponding behavior (input and output), with the effect of noise applied.

In the interface mentioned above, an `EQ` can never return "Yes" as in MAT. This work is left to the Reviser, that will halt the learning process according to the termination criterion chosen (see Section 3.2). The Prune signal does not require us to modify the code of a MAT Learner, as it can be implemented by restarting the Learner without requiring further access to the Learner's internals. The Reviser's caching of observations ensures that this operation does not add to the query complexity of the process.

Remark 3. *The main cost of learning comes from* unit interactions with the system *— each individual symbol that is inputted into or outputted by the system — as these tests are generally costly to perform and that cost cannot be compensated.*

3.2 The Reviser

The Reviser agent is the core of the C$\mathcal{E}$AL framework. It concretizes its main idea: taking the storage and handling of observations *out* of the Learner's prerogatives. Its task is to update the observation tree $\mathscr{T}$ on which a Learner is trained. We will assume the following interface is made available to the Reviser:

Definition 1 (Operations on observation trees)**.** *Given an observation tree $\mathscr{T}$, we define the following functions to access and modify $\mathscr{T}$:*

- $\textsc{Lookup}_{\mathscr{T}} : \Sigma^* \to (\Gamma^* \cup \{\textsc{Null}\})$ *receives an input word i and returns output o if (i,o) is present in the observation tree $\mathscr{T}$. Otherwise,* $\textsc{Null}$ *is returned.*
- $\textsc{Update}_{\mathscr{T}} : (\Sigma^* \times \Gamma^*) \to 2$ *updates the observation tree $\mathscr{T}$ to take into account a new query pair, revoking conflicting information if necessary. Returns $\top$ if the new information conflicts with $\mathscr{T}$.*

Note that the function $\textsc{Update}$ is the only one that alters the tree and handles conflicts. Implementations of these functions are given in [16, Appendix A]. Using these functions, we can now define the *language* of an observation tree as the set of observations that it can transmit to a Learner, and provide a formal definition of a conflict as a non-additive change to the language of $\mathscr{T}$.

Definition 2. *Given an observation tree $\mathscr{T}$, we call* language *of $\mathscr{T}$ the following set*

$$\mathscr{L}_{\mathscr{T}} = \{(i, \textsc{Lookup}_{\mathscr{T}}(i)) \in \Sigma^* \times \Gamma^* \mid \textsc{Lookup}_{\mathscr{T}}(i) \in \Gamma^*\} .$$

Definition 3. *An observation (i,o)* conflicts *with an observation tree $\mathscr{T}$ when the tree $\mathscr{U}$ obtained by calling $\textsc{Update}_{\mathscr{T}}(i,o)$ satisfies $\exists (i',o') \in \mathscr{L}_{\mathscr{T}},\ o' \neq \textsc{Lookup}_{\mathscr{U}}(i')$. Two observations (i,o) and (i',o')* conflict*, written $(i,o) \lightning (i',o')$, when there is an input word $i'' \in \mathsf{prefixes}(i) \cap \mathsf{prefixes}(i')$ such that $o[|i''|] \neq o'[|i''|]$.*

Note that a conflict appears not between the System and the Reviser, but signifies that the Reviser wants to update its answer to some information previously given to the Learner.

Definition 4 (Reviser)**.** *The Reviser contains an observation tree $\mathscr{T}$ and implements four operations:*

① $\textsc{Apply}_{\mathscr{T}} : (\Sigma^* \times \Gamma^*) \to (\Gamma^* \cup \{\texttt{Prune}\})$ *updates $\mathscr{T}$ with the observation gained from a system test. It then either returns the query output or prunes the learner if a conflict is detected.*

② $\textsc{Read}_{\mathscr{T}} : \Sigma^* \to (\Gamma^* \cup \{\texttt{Prune}\})$ *looks in $\mathscr{T}$ for a query answer and either returns it or tests the system if necessary. Note that if a test is performed, then $\mathscr{T}$ is updated accordingly.*

```
Algorithm 1: $\textsc{Apply}_{\mathscr{T}}(i,o)$
  Data: $(i,o)$ trace from the SUL.
  if $\textsc{Update}_{\mathscr{T}}(i,o)$ then
      return Prune;
  return $o$;
```

```
Algorithm 2: $\textsc{Read}_{\mathscr{T}}(i)$
  Data: The queried string i.
  $o \leftarrow \textsc{Lookup}_{\mathscr{T}}(i)$;
  if $o \neq \textsc{Null}$ then return $o$ ;
  return $\textsc{Apply}_{\mathscr{T}}(\texttt{System}(i))$;
```

(3) $\text{CHECK}_{\mathcal{T}}$: Mealy $\rightarrow ((\Sigma^* \times \Gamma^*) \cup \{\text{NULL}\})$ *performs a consistency check of a given Mealy machine hypothesis against the observation tree $\mathcal{T}$. Returns a counterexample if found or* NULL *if no divergences are found.*

(4) $\text{TEST}_{\mathcal{T}}$: Mealy $\rightarrow ((\Sigma^* \times \Gamma^*) \cup \{\texttt{Prune}\})$ *is the function used to look for counterexamples in the System. It takes a hypothesis proposed by the learner and coherent with the observation tree, and tests the SUL until a counterexample or a conflict is found. The tests are taken from* `sampleWord` *which is instantiated by an off-the-shelf test suite generating algorithm (e.g. the Wp-method [20] or Hybrid-ADS [26,29]) in practice.*

Algorithm 3: $\text{CHECK}_{\mathcal{T}}(\mathcal{H})$

```
Data: Hypothesis $\mathcal{H}$
for $(i,o) \in \mathcal{T}$ do
    if $\mathcal{H}^*(i) \neq o$ then
        return $(i,o)$;
return NULL;
```

Algorithm 4: $\text{TEST}_{\mathcal{T}}(\mathcal{H})$

```
Data: A hypothesis $\mathcal{H}$ coherent with $\mathcal{T}$.
while $\top$ do
    $w \leftarrow$ sampleWord();
    $(i,o) \leftarrow$ System($w$);
    if $\text{APPLY}_{\mathcal{T}}(i,o) =$ Prune then return Prune ;
    if $\mathcal{H}(i) \neq o$ then return (i,o) ;
```

Crucially, the above functions rely on the observation tree's interface to handle the conflict as they arise, forwarding the `Prune` command to the Learner when needed.

Update Strategies At the core of dealing with conflicts is the idea of identifying information that will be sacrificed for the sake of cohesion. The way this is achieved depends largely on the type of conflict, and the *meaning* of observing such a conflict. We propose two ways to resolving conflicts in C$\mathcal{E}$AL:

(1) `Most Recent`: When a conflict is identified, the most recently observed (freshest) query information is committed to the observation tree, and the previous one suppressed, if needed. This approach makes sense, for example if the target system has mutated and we are only interested in capturing the most up-to-date behavior, or as a base default strategy. We define $\text{prefixes}(i,o) = \{(i[1,n], o[1,n]) \mid 0 \leq n \leq |i|\}$.

Proposition 1. *In the case of the* `Most Recent` *update strategy, given a stream of tests $((i_k,o_k)_{k\in\mathbb{N}})$, at any step $K \in \mathbb{N}$: $\mathcal{L}_T = \{\text{prefixes}(i_k,o_k) \mid 0 \leq k \leq K \wedge \nexists k < l \leq K,$ s.t. $(i_k,o_k) \lightning (i_l,o_l)\}$.*

Example 2. *In Example 1, the right-hand tree is the result of observing (aaa, abb) starting from the left-hand tree. Notice that the sets prefixes of the sets of observations in Example 1 verify Proposition 1.*

(2) `Most Frequent`: When two possible output sequences conflict for a given input sequence, the most frequently observed one is returned to the Learner. This information can be obtained passively by keeping track of naturally occurring repetitions of queries, or actively by specifying a sample size on which the frequency is estimated. This approach makes sense for example for conflicts that are due to unwanted statistical noise in the observations.

We define $\text{Count}(i,o) = |\{k \mid (i,o) \in \mathcal{P}(\text{prefixes})(\{(i_k,o_k)_{k\in K}\})\}|$ as the number of observations of which (i,o) is a prefix in an observation stream $(i_k,o_k)_{k\in\mathbb{N}}$ considered at step K.

Proposition 2. *In the case of the* `Most Frequent` *update strategy, given a stream of tests $((i_k,o_k)_{k\in\mathbb{N}})$, at any step $K \in \mathbb{N}$:*

$$mf((i_k,o_k),(i_l,o_l)) \triangleq \text{Count}(i_k,o_k) < \text{Count}(i_l,o_l) \ \vee \ (\text{Count}(i_k,o_k) = \text{Count}(i_l,o_l) \ \wedge \ k < l)$$

$$\mathcal{L}_{\mathcal{T}} = \{\text{prefixes}(i_k,o_k) \mid k \leq K \wedge \ \nexists k < l \leq K, \text{ s.t. } (i_k,o_k) \lightning (i_l,o_l) \ \wedge mf((i_k,o_k),(i_l,o_l))\}$$

We present implementations of UPDATE and LOOKUP fitting these two strategies in [16, Appendix A], and proofs of the above properties in [16, Appendix B].

Remark 4. *An observation (i,o) conflicting with an observation tree $\mathscr{T}$ implies that $(i,o) \lightning (i',o')$ for some $(i',o') \in \mathscr{L}_{\mathscr{T}}$. The other implication is not always true, e.g. for the `Most Frequent` update strategy.*

Termination The termination criteria of C𝓔AL are the same as those used in MAT in practice: in our experiment, we terminate when our currently selected hypothesis has survived for a fixed number of tests that is deemed sufficient, or if a predefined limit number of queries is reached.

Hypothesis Selection Active automata learning involves the production of a sequence of hypotheses that are refined over time, with the goal of converging towards a correct one. As such, a key characteristic of different approaches to automata learning is how a final model is to be selected, out of the many hypotheses. In the case of MAT this is simple: Learning produces a sequence of ever more accurate models, until termination occurs with a positive equivalence query. It is then logical to pick the most recently produced hypothesis as the final model. However, when dealing with conflicts and different update strategies, this is no longer necessarily the case for C𝓔AL. In particular, when it comes to electing a model out of a sequence of hypothesis, C𝓔AL has two options:

- `Most Recent`: This hypothesis selection strategy is the one known classically: the most recently produced hypothesis is the one to be elected as final. This strategy is sensible in the case of learning with no noise, or in the case of learning targets that evolve over time.
- `Most Frequent`: In this selection strategy, the sequence of hypotheses is analyzed to elect a final model. We count the frequency of each unique model (up to language equivalence) over the sequence, and elect the most frequently occurring one. This strategy makes sense when dealing with noise, as we may be producing (rarely) hypotheses that capture noisy behavior that is fixed over time. As such, we want to select not the latest model produced, but the one that is the most stable. This strategy can be implemented efficiently in practice (using hash fingerprints and counters, for example) and on-the-fly during learning, allowing us to not have to store the whole sequence of hypotheses as it is produced.

3.3 Interface Implementation

We now explain how to build the interface described in Sec. 3.1 using the Reviser. This mostly amounts to implementing membership and equivalence queries, as the testing interface is simply composed of calls to `System`. **Membership** queries can be defined, for $i \in \Sigma^*$, as $\mathtt{MQ}(i) = \textsc{Read}_{\mathscr{T}}(i)$. When $\mathscr{T}$ does not have the answer to this particular query, $\textsc{Read}_{\mathscr{T}}$ sends it through to the SUL (with the call to `System`) and the result is applied in $\mathscr{T}$. This process is illustrated in Fig. 4.

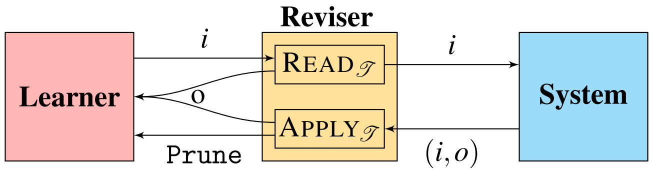


Figure 4: Implementation of Membership Queries (MQ) in the Reviser.

Equivalence queries are handled in two steps. First, the hypothesis given by the learner is checked against the observation tree using $\textsc{Check}_{\mathscr{T}}$. If a counterexample is found, it is returned. Otherwise, the

Reviser tests the System to discover new information and update the tree. If a counterexample is found, either it is returned to the learner or, if a conflict arose, the learner is pruned. (Fig. 5).

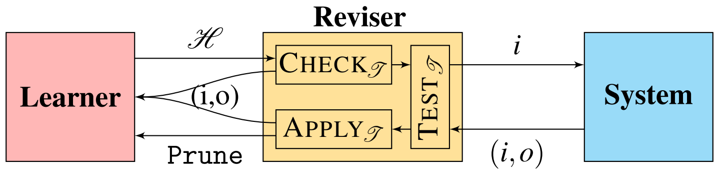


Figure 5: Implementation of Equivalence Queries (EQ) in the Reviser.

Remark 5 (Modularity)**.** *We present* C𝓔AL *in the case of* black-box *learning where we do not have access to any information from the system but can interact with it. However, the framework is fully modular, as can be seen from the high-level functions presented in Section 3.2: one can interface any model-checking (or other) method before the calls to system in* TEST$_{\mathscr{T}}$ *and* READ$_{\mathscr{T}}$ *when related models (specifications, parts of the System. . .) are available, allowing* C𝓔AL *to perform gray-box or even white-box learning.* C𝓔AL *focuses on the* storage *of information, without restrictions on its acquisition.*

Correctness Proposition 1 and Proposition 2 characterize the language of the Reviser, and the following results describe its interactions with the Learner and the System (as proved in [16, Appendix B]).

Lemma 1. *During an execution of* C𝓔AL*, all tests on the System are integrated in $\mathscr{T}$ through* UPDATE$_{\mathscr{T}}$*, and the Learner queries are answered according to $\mathscr{L}_{\mathscr{T}}$.*

Proposition 3. `Prune` *is sent to the Learner exactly when a new observation conflicts with $\mathscr{T}$.*

3.4 Optimizations

C𝓔AL allows us to implement two main optimizations, both in the framework and its algorithms.

Query Caching One of the direct benefits of the presence of the Reviser is that the learner does not have to cache membership queries to avoid repeating them, as it obtains knowledge through the Reviser's data structure. It especially offers a good basis to discuss algorithms that are based on the observation tree themselves [30, 33], for which the Learner's data structure is very limited.

Specialized Pruning In order to fully support the simplistic interface of a classic MAT learner, we restart it — at no extra query cost (see Section 3.1) — when the `Prune` signal is sent. For specific Learner data structures, the time-complexity of this operation can be enhanced by suppressing only the part of the data structure that is impacted by the conflict (instead of restarting it completely). This optimization, however, will be specific to each learning algorithm. In the same way as the Learner can read the Reviser' observation tree, a C𝓔AL-specific Learner could compute its internal data-structure directly from it after a pruning without requiring restarts.

4 Evaluation

We introduced C𝓔AL to extend the power of classic MAT learners into environments that may cause conflicts, for instance caused by *noise*. In practice, each symbol inputted in or outputted by the SUL can be noisy, making longer queries more likely to have noisy results overall. Recall that poorly guessing the number n of query repetitions can lead to a learning failure (if noise is integrated in the system), or a timeout (if too many repetitions are made). Hence noise does not only affect the *efficiency*, but also the *success rate* of the learning, i.e., the proportion of runs that end with a correct hypothesis.

It is then important to measure how well C$\mathcal{E}$AL can avert these negative effects. In particular, we are interested in testing if C$\mathcal{E}$AL's approach is sufficient to allow for an improvement of learning environments plagued by noise[1]. We evaluate this through the following research question:

Across different realistic model learning targets, types of noise, and noise levels, does C$\mathcal{E}$AL provide a better learning environment in terms of both success rates and number of tests issued when compared to the state-of-the-art MAT-based approaches?

4.1 Experiments

We first present the experimental setup and the outlines of the experimental results. We focus on the difference between *uncontrollable parameters*, i.e., that are part of the target and its environment and can't be altered by the learning setup, and the *controllable* ones, that are chosen when designing a learning session. *Uncontrollable parameters* are related to the SUL and its environment.

Realistic Targets We run our experiments over a range of 36 Mealy machines representing real world systems from previous successful model learning applications [27]. These range in size between 4 and 66 states, with alphabet sizes between 7 and 22 input symbols.

Different Types of Noise We run the targets on different types of realistic simulated noise, namely input and output noise as described above.

Different Levels of Noise We run the above mentioned noise types over 3 different levels: 0.01, 0.05, and 0.1. These indicate the probability that each symbol has of being noisy.

Given the above constraints, we aim to reach the best performing learning session by manipulating the following *controllable parameters*:

Framework We run each experiment under both MAT and C$\mathcal{E}$AL.

Algorithm We run each experiment under `L#`, `TTT`, `KV`, and $\mathtt{L}^\star$. We use the implementations of these algorithms provided in LearnLib [23]. Notably, $\mathtt{L}^\star$ is implemented with Rivest & Schapire's improvements [28] and we re-implemented `L#` completely to incorporate it into the LearnLib library.

Number of Repeats We compare different numbers of repeats used in majority voting test results to remove noise in MAT, or in sampling frequencies for the update strategy in C$\mathcal{E}$AL. We use one of 3 different levels of repeats, in pairs of (*min_repeats*,*max_repeats*): $(5,10),(10,20),(20,30)$[2].

Each experiment uses the following settings to enhance its learning, independent of the above mentioned variables. Firstly, caching of previously observed queries is done wherever possible in MAT, and the `Most Recent` update and `Most Frequent` hypothesis selection strategies are used in C$\mathcal{E}$AL, for simplicity. Secondly, the Hybrid-ADS [26] equivalence testing algorithm is used for all runs as we found it to be the best performing for our experiments. Thirdly, each independent experiment is performed with 100 runs, and its results averaged for consistency. And finally, each run is allowed to use up to 10 million queries before an unsuccessful timeout is declared.

[1]We compare success rates and number of tests issued instead of running times as to make hardware-agnostic benchmarks that capture the main factors in both efficiency and correctness.

[2]Each test is repeated *min_repeats* times and then if at least 80% of the queries agree the result is returned. Else, the query is repeated until it is the case or *max_repeats* is reached at which point the majority answer is returned.

Due to the vast number of variables considered, we are unable to fully describe the result of the over 18,000 distinct experiments, and close to 2 million runs that we have performed. However this is not required to rigorously answer our research question. What we have to consider is, for each combination of our independent variables (target, noise type, and noise level), which framework allows for the most efficient learning configuration.

The graphs below (Figs. 6 - 11) summarize, for each target and for all levels and types of noise, the success rates and number of symbols tested of the best controllable parameter profile for both MAT and C$\mathcal{E}$AL. We have also included all the data used, as well as conclusions in [16, Appendix C].

4.2 Analysis

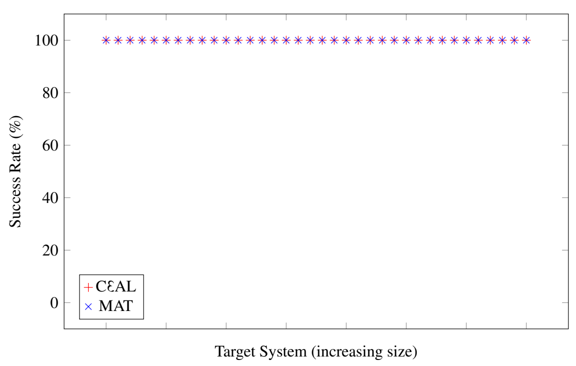
We now analyze the results of these experiments to draw some high-level conclusions about how MAT and C$\mathcal{E}$AL compare and answer our research question.

Success Rates First and foremost, we discuss the impact of different parameters on the success rates of the experiments. We can see from the graphs (Figs. 6a - 11a) that, as expected, while the exact type of noise does not have a significant impact on success rates and test counts, the *level of noise* does. In particular, at a very low level of 0.01% (Figs. 6a, 9a) both frameworks are capable of maintaining perfect success rates. However, once the level increases to 0.05% (Figs. 7a, 10a), MAT's success rate starts to fluctuate, more so in bigger targets. C$\mathcal{E}$AL too seems to be slightly affected by an increase in noise, but overall maintains a success rate close to 100%. Once the noise is increased to its highest level, 0.1% (Figs. 8a, 11a), we can see that MAT's success rates reduce significantly, while C$\mathcal{E}$AL's tend to stay high for a great number of targets, until they inevitably decrease when faced with massive targets at this level of noise. C$\mathcal{E}$AL **manages to stay consistently reliable in the face of these large alphabets**.

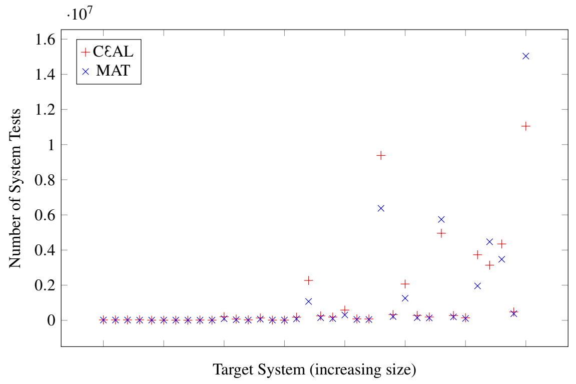
Efficiency Let us now turn our attention to the system test count graphs (Figs. 6b - 11b). Overall we see an expected picture: Larger systems require more tests to be learned. A particular caveat to notice however, is that while MAT appears to have quite efficient runs on large noisy targets, their respective success rates are considerably lower. Although efficiency of learning is certainly important, it is of low use if at the end the reported hypothesis is not correct. This result is expected: If a learning run fails due to, for example, high noise not being fully filtered out, the MAT learner will collapse before it finishes running. This leaves a final test count that is quite low, but also gives us an incorrect hypothesis.

Overall Results Perhaps most importantly, C$\mathcal{E}$AL **provides the most efficient *correct* configuration in 70% of the experiments**, having a better success rate than MAT or the same with a lower average number of tests used. We provide this result for each individual experiment in [16, Appendix C]. In particular, in every experiment C$\mathcal{E}$AL performs with a success rate that is at least as high as MAT's, often outperforming it. In addition to this, experiments ran with C$\mathcal{E}$AL had an overall success rate of 95.5% compared to MAT's 79.5% success rate. This alone has allowed C$\mathcal{E}$AL to perform successful runs that no configuration of MAT was able to perform, namely learning moderate to large targets at 0.1% noise.

We found that a lot of the improvements provided by C$\mathcal{E}$AL are commonly a consequence of it being more successful when running at a *lower number of repeats* when compared to the ones required by MAT. This solidifies our initial hypothesis of there being a benefit in reducing the number of repeats used when learning noisy targets. The above provides enough supporting evidence to answer our research question positively: C$\mathcal{E}$AL **provides a better learning environment in terms of both success rates and number of tests issued when compared to the state-of-the-art MAT-based approaches**.

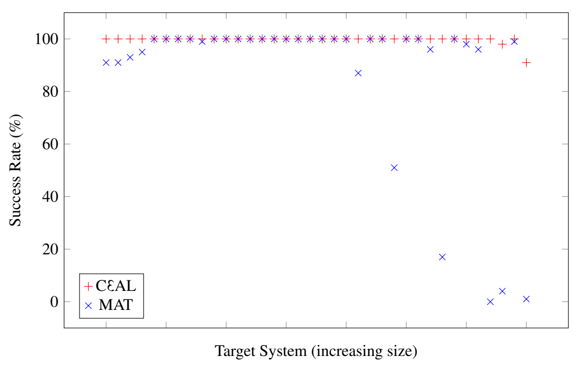


(a) Success rate (higher is better)

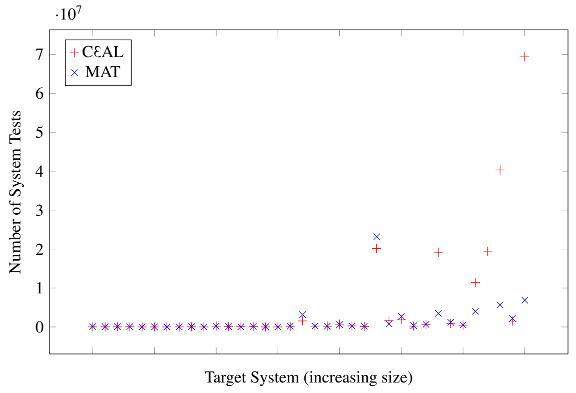


(b) Average test use (lower is better)

Figure 6: **INPUT** noise at **0.01%**.

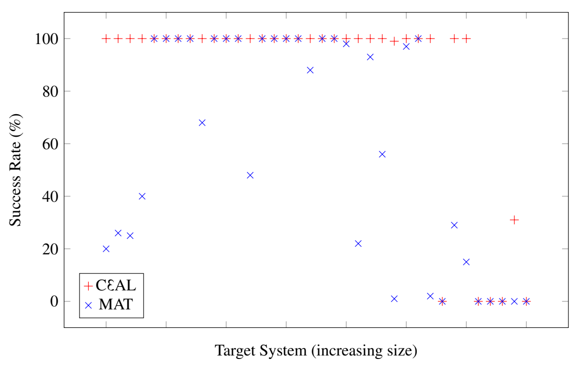


(a) Success rate (higher is better)

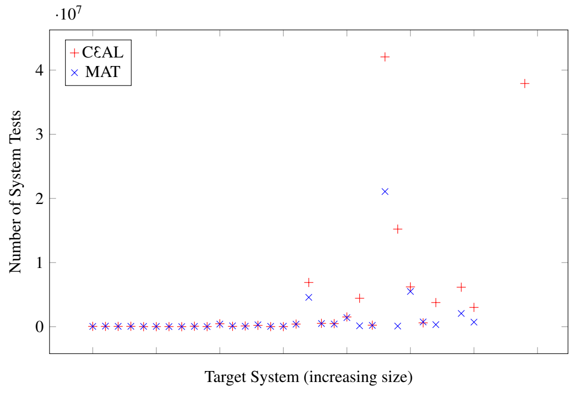


(b) Average test use (lower is better)

Figure 7: **INPUT** noise at **0.05%**.

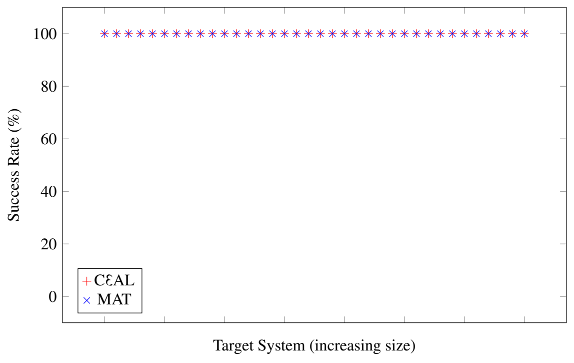


(a) Success rate (higher is better)

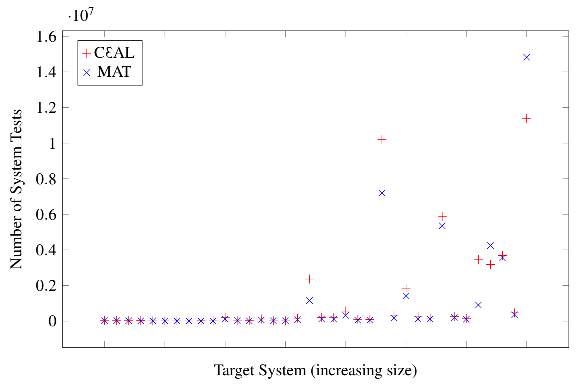


(b) Average test use (lower is better)

Figure 8: **INPUT** noise at **0.1%**.

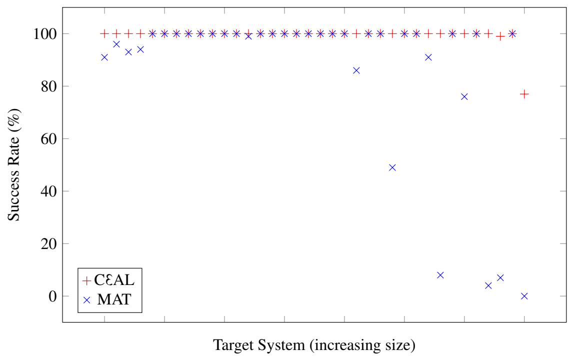


(a) Success rate (higher is better)

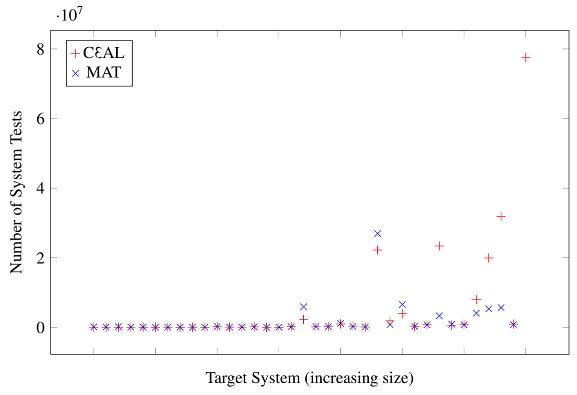


(b) Average test use (lower is better)

Figure 9: **OUTPUT** noise at **0.01%**.

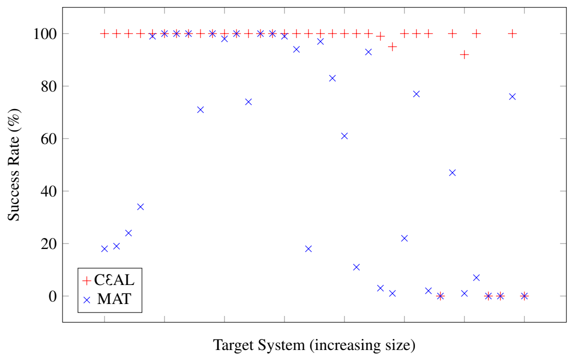


(a) Success rate (higher is better)

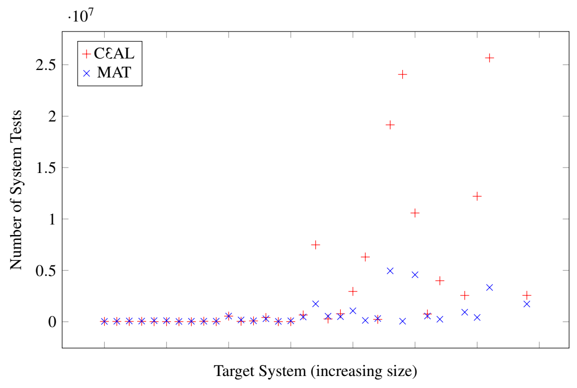


(b) Average test use (lower is better)

Figure 10: **OUTPUT** noise at **0.05%**.



(a) Success rate (higher is better)



(b) Average test use (lower is better)

Figure 11: **OUTPUT** noise at **0.1%**.

Other Findings We report on other interesting findings in [16, Appendix D]. Two results, however, are of particular significance: As already accepted by the community [5], we can confirm that indeed most of the tests spent in learning are used to realize equivalence queries. In particular, we found that **equivalence tests account for 89.1% of tests in MAT, 59.8% in $\mathbb{C}\mathcal{E}$AL, and 66.3% on average**.

Perhaps more surprisingly, the commonly accepted ordering of Learner efficiencies did not surface in our experiments. From theoretically most performant to least we have $\mathtt{L}^{\#}$, `TTT`, `KV`, and $\mathtt{L}^{\star}$, based on complexity analyses in MAT. Through our experiments we have found that, at least in our particular case of black-box learning (i.e. learning from SUL tests only) of Mealy machines with noise and randomized testing algorithms, this ordering cannot be seen in the results. The best configurations for each framework were not consistent on which algorithm performed the best. Not only was there no clear ”winning” algorithm, we found no pattern based on noise, target and alphabet size, or number of repeats that had a strong enough correlation to the better performance of any one algorithm.

We believe that this is not a flaw of the complexity analyses themselves. It is simply that complexity analyses in MAT abstract away the biggest cost of learning: equivalence tests. It may be that more recent algorithms have a theoretical (and membership query based) advantage over classic algorithms, however the nature of randomized equivalence oracles seems to be a bigger agent of chaos, and a good or bad run of the equivalence oracle quickly overshadows the small advantage that some algorithms may have.

5 Related Work

There has been extensive work on finding ways of applying classic learning algorithms like $\mathtt{L}^{\star}$ [4], `KV` [24], `TTT` [22], and $\mathtt{L}^{\#}$ [33] to real world systems such as passports [2], network protocol implementations [15, 18, 19], and bank cards [1]. All these works rely on ad-hoc implementations of noise handling which is inefficient and not formalized in the MAT framework. One of the goals of our framework, which replaces the teacher with the SUL and the Reviser, is to discuss how noise can be dealt with in the learning process, independently of the type of Learner being used. The LearnLib library [23] provides *caches* that can be placed in the learning environment to avoid the repetition of queries, much like observation trees. Note that our Reviser agent goes further than LearnLib as it provides the ability to act as membership and equivalence oracles, test the system, and act on conflicts by pruning the learner's data structure in an efficient (query-wise) and correct manner.

There has also been previous work in improving the efficiency of model learning strategies for mutating targets by reusing previously learned behavior, using *adaptive* learning algorithms [11,14,17,21,35]. These algorithms work by being able to start learning with pre-seeded information of previous runs that has been confirmed to still apply in the current target, or by being able to filter this information themselves if it is found to no longer apply. Additionally, there has been some work on *Lifelong Learning* [7], where model learning and model checking are used together to run over the development lifecycle of a system. This allows for the quick discovery of bugs in the development cycle. However, when these are found and corrected, learning needs to be *manually restarted*.

Our model learning framework improves on these two lines of work, being able to autonomously correct itself when faced with conflicts. It can do so without any notification of mutations in the system, allowing it to be applied to complete closed-box systems, unlike the current state-of-the-art adaptive algorithm [17]. Additionally, it is capable of continuously checking for changes in the system, much like Lifelong Learning, but requiring no human interaction on system changes. These characteristics make it resilient to real world noise, allowing the learner to correct itself as it identifies the correct behavior.

Our work participates in the current trend trying to link learning to testing, which spans communities, e.g. formal approaches [3], genetic approaches [31], and fuzzing [36]. In this context, CℰAL provides a modular framework upon which other techniques can be added. In active model learning, this trend also matches the interest in observation-tree based algorithms [30, 33], which we instantiate in CℰAL. The role of observation structures in learning and testing is a long-standing lore [8] that can be leveraged to enhance the learning approach and its modularity with testing methods.

6 Conclusion and Future Work

This paper explores efficient ways to handle conflicts during active learning. We build on the idea that recovering from conflicts is best done by splitting information collection and the construction of the Learner's data structure, two operations that are conflated in MAT.

We introduce the Conflict Aware Active Automata Learning (CℰAL) framework as an alternative to MAT. CℰAL directly represents the SUL and introduces a *Reviser* tasked with testing it, storing and curating the observations. CℰAL provides a way to accept *some* conflicts to reach the Learner and to recover from them without requiring to test the SUL anew.

To test the efficiency of CℰAL, we conducted a large body of experiments on real targets using several state-of-the-art algorithms. We found that **not only does** CℰAL **always improves on MAT in terms of success rates**, obtaining an overall **success rate of 95,5% against MAT's 79,5%** it most importantly **enables the learning of previously un-learnable SULs**, typically complex systems plagued with a high level of noise. Our experiments further put into light the impact of equivalence tests, both in terms of variability of the results and sheer cost, with an average of **66.3% of testing cost spent on equivalence**.

In the future, we would like to explore the use and design of testing algorithms for active learning, as their efficiency seems to be able to overshadow the difference between learning algorithms. CℰAL's modular nature also allows us to seamlessly build a *gray-box* environment i.e. to gain information from different sources in the Reviser (e.g. specifications, access to source code). This would offset the cost of equivalence queries by using cheaper sources of observations when searching for counterexamples.

Assessing the efficiency of CℰAL on a real case of mutating targets would be of interest, as an evaluation, as an opportunity to fine-tune the framework for such task, and as a demonstration of the improved reach of active learning. Similarly, testing the `Most Frequent` update strategy in practice against high noise levels would be of interest.

Acknowledgements

This work was partially supported by the EPSRC Standard Grant CLeVer (EP/S028641/1), ERC grant Autoprobe (no. 101002697), and a Royal Society Wolfson Fellowship. The authors would like to also thank Denis Timm and the TSG and HPC groups at UCL Computer Science department for their support in running the experiments in this paper.

References

[1] F. Aarts, J. De Ruiter, and E. Poll. Formal Models of Bank Cards for Free. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 461–468, March 2013. `doi:10.1109/ICSTW.2013.60`.

[2] Fides Aarts, Julien Schmaltz, and Frits Vaandrager. Inference and Abstraction of the Biometric Passport. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification, and Validation*, Lecture Notes in Computer Science, pages 673–686, Berlin, Heidelberg, 2010. Springer. `doi:10.1007/978-3-642-16558-0_54`.

[3] Bernhard K. Aichernig, Wojciech Mostowski, Mohammad Reza Mousavi, Martin Tappler, and Masoumeh Taromirad. *Model Learning and Model-Based Testing*, pages 74 – 100. Lecture Notes in Computer Science. Springer Nature, 7 2018. `doi:10.1007/978-3-319-96562-8_3`.

[4] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, November 1987. URL: `http://www.sciencedirect.com/science/article/pii/0890540187900526`, `doi:10.1016/0890-5401(87)90052-6`.

[5] Kousar Aslam, Loek Cleophas, Ramon Schiffelers, and Mark van den Brand. Interface protocol inference to aid understanding legacy software components. *Software and Systems Modeling*, 19(6):1519–1540, November 2020. `doi:10.1007/s10270-020-00809-2`.

[6] John Backes, Byron Cook, Andrew Gacek, Neha Rungta, and Michael W. Whalen. One-click formal methods. In *ICST 2020*, 2019. URL: `https://www.amazon.science/publications/one-click-formal-methods`.

[7] Alexander Bainczyk, Bernhard Steffen, and Falk Howar. Lifelong Learning of Reactive Systems in Practice. In Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, and Einar Broch Johnsen, editors, *The Logic of Software. A Tasting Menu of Formal Methods*, volume 13360, pages 38–53. Springer International Publishing, Cham, 2022. Series Title: Lecture Notes in Computer Science. `doi:10.1007/978-3-031-08166-8_3`.

[8] Therese Berg, Olga Grinchtein, Bengt Jonsson, Martin Leucker, Harald Raffelt, and Bernhard Steffen. On the correspondence between conformance testing and regular inference. In Maura Cerioli, editor, *Fundamental Approaches to Software Engineering*, pages 175–189, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. `doi:978-3-540-31984-9_14`.

[9] James Bornholt, Rajeev Joshi, Vytautas Astrauskas, Brendan Cully, Bernhard Kragl, Seth Markle, Kyle Sauri, Drew Schleit, Grant Slatton, Serdar Tasiran, Jacob Van Geffen, and Andrew Warfield. Using Lightweight Formal Methods to Validate a Key-Value Storage Node in Amazon S3. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles CD-ROM*, pages 836–850, Virtual Event Germany, October 2021. ACM. `doi:10.1145/3477132.3483540`.

[10] Quentin Carbonneaux, Noam Zilberstein, Christoph Klee, Peter W. O'Hearn, and Francesco Zappa Nardelli. Applying formal verification to microkernel IPC at meta. In Andrei Popescu and Steve Zdancewic, editors, *CPP '22: 11th ACM SIGPLAN International Conference on Certified Programs and Proofs, Philadelphia, PA, USA, January 17 - 18, 2022*, pages 116–129. ACM, 2022. `doi:10.1145/3497775.3503681`.

[11] Sagar Chaki, Edmund Clarke, Natasha Sharygina, and Nishant Sinha. Verification of evolving software via component substitutability analysis. *Formal Methods in System Design*, 32(3):235–266, June 2008. `doi:10.1007/s10703-008-0053-x`.

[12] Andrey Chudnov, Nathan Collins, Byron Cook, Joey Dodds, Brian Huffman, Colm MacCárthaigh, Stephen Magill, Eric Mertens, Eric Mullen, Serdar Tasiran, Aaron Tomb, and Eddy Westbrook. Continuous formal verification of amazon s2n. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification*, pages 430–446, Cham, 2018. Springer International Publishing. `doi:10.1007/9783319961422_26`.

[13] Byron Cook, Kareem Khazem, Daniel Kroening, Serdar Tasiran, Michael Tautschnig, and Mark R. Tuttle. Model checking boot code from aws data centers. In *CAV 2018*, 2018. URL: `https://www.amazon.science/publications/model-checking-boot-code-from-aws-data-centers`.

[14] Carlos Diego Nascimento Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simão. Learning to Reuse: Adaptive Model Learning for Evolving Systems. In *IFM*, volume 11918 of *LNCS*, pages 138–156. Springer, 2019. doi:10.1007/978-3-030-34968-4_8.

[15] Tiago Ferreira, Harrison Brewton, Loris D'Antoni, and Alexandra Silva. Prognosis: closed-box analysis of network protocol implementations. In *SIGCOMM*, pages 762–774. ACM, 2021. doi:10.1145/3452296.3472938.

[16] Tiago Ferreira, Léo Henry, Raquel Fernandes da Silva, and Alexandra Silva. Conflict-aware active automata learning, 2023. arXiv:2308.14781.

[17] Tiago Ferreira, Gerco van Heerdt, and Alexandra Silva. *Tree-Based Adaptive Model Learning*, pages 164–179. Springer Nature Switzerland, Cham, 2022. doi:10.1007/978-3-031-15629-8_10.

[18] Paul Fiterau-Brostean, Ramon Janssen, and Frits W. Vaandrager. Combining Model Learning and Model Checking to Analyze TCP Implementations. In *CAV*, volume 9780 of *LNCS*, pages 454–471. Springer, 2016. doi:10.1007/978-3-319-41540-6_25.

[19] Paul Fiterau-Brostean, Toon Lenaerts, Erik Poll, Joeri de Ruiter, Frits W. Vaandrager, and Patrick Verleg. Model learning and model checking of SSH implementations. In *SPIN*, pages 142–151. ACM, 2017. doi:10.1145/3092282.3092289.

[20] S. Fujiwara, G. v. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi. Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603, 1991. doi:10.1109/32.87284.

[21] David Huistra, Jeroen Meijer, and Jaco van de Pol. Adaptive Learning for Learn-Based Regression Testing. In Falk Howar and Jiří Barnat, editors, *Formal Methods for Industrial Critical Systems*, volume 11119, pages 162–177. Springer International Publishing, Cham, 2018. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-030-00244-2_11.

[22] Malte Isberner, Falk Howar, and Bernhard Steffen. The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning. In Borzoo Bonakdarpour and Scott A. Smolka, editors, *Runtime Verification*, volume 8734, pages 307–322. Springer International Publishing, Cham, 2014. Series Title: Lecture Notes in Computer Science. doi:10.1007/978-3-319-11164-3_26.

[23] Malte Isberner, Falk Howar, and Bernhard Steffen. The Open-Source LearnLib. In *CAV*, volume 9206 of *LNCS*, pages 487–495, 2015. doi:10.1007/978-3-319-21690-4_32.

[24] Michael J. Kearns and Umesh Virkumar Vazirani. *An introduction to computational learning theory*. MIT Press, Cambridge, Mass, 1994. doi:10.7551/mitpress/3897.001.0001.

[25] Quang Loc Le, Azalea Raad, Jules Villard, Josh Berdine, Derek Dreyer, and Peter W. O'Hearn. Finding real bugs in big programs with incorrectness logic. *Proc. ACM Program. Lang.*, 6(OOPSLA1), apr 2022. doi:10.1145/3527325.

[26] D. Lee and M. Yannakakis. Testing finite-state machines: state identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994. doi:10.1109/12.272431.

[27] Daniel Neider, Rick Smetsers, Frits Vaandrager, and Harco Kuppens. Benchmarks for Automata Learning and Conformance Testing. In Tiziana Margaria, Susanne Graf, and Kim G. Larsen, editors, *Models, Mindsets, Meta: The What, the How, and the Why Not? Essays Dedicated to Bernhard Steffen on the Occasion of His 60th Birthday*, pages 390–416. Springer International Publishing, Cham, 2019. doi:10.1007/978-3-030-22348-9_23.

[28] Ronald L. Rivest and Robert E. Schapire. Inference of Finite Automata Using Homing Sequences. *Inf. Comput.*, 103:299–347, 1993. doi:10.1006/inco.1993.1021.

[29] Wouter Smeenk, Joshua Moerman, Frits Vaandrager, and David N. Jansen. Applying automata learning to embedded control software. In Michael Butler, Sylvain Conchon, and Fatiha Zaïdi, editors, *Formal Methods and Software Engineering*, pages 67–83, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-25423-4_5.

[30] Michal Soucha and Kirill Bogdanov. Observation Tree Approach: Active Learning Relying on Testing. *The Computer Journal*, 63(9):1298–1310, 07 2019. doi:10.1093/comjnl/bxz056.

[31] Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. Time to learn - learning timed automata from tests. In Étienne André and Mariëlle Stoelinga, editors, *Formal Modeling and Analysis of Timed Systems - 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27-29, 2019, Proceedings*, volume 11750 of *Lecture Notes in Computer Science*, pages 216–235. Springer, 2019. `doi:10.1007/978-3-030-29662-9\_13`.

[32] Frits W. Vaandrager. Model learning. *Commun. ACM*, 60:86–95, 2017. `doi:10.1145/2967606`.

[33] Frits W. Vaandrager, Bharat Garhewal, Jurriaan Rot, and Thorsten Wißmann. A New Approach for Active Automata Learning Based on Apartness. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 223–243. Springer, 2022. `doi:10.1007/978-3-030-99524-9_12`.

[34] Pepe Vila, Pierre Ganty, Marco Guarnieri, and Boris Köpf. CacheQuery: Learning Replacement Policies from Hardware Caches. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, pages 519–532, New York, NY, USA, 2020. Association for Computing Machinery. event-place: London, UK. `doi:10.1145/3385412.3386008`.

[35] Stephan Windmüller, Johannes Neubauer, Bernhard Steffen, Falk Howar, and Oliver Bauer. Active Continuous Quality Control. In *Proceedings of the 16th International ACM Sigsoft Symposium on Component-Based Software Engineering*, CBSE '13, pages 111–120, New York, NY, USA, 2013. Association for Computing Machinery. event-place: Vancouver, British Columbia, Canada. `doi:10.1145/2465449.2465469`.

[36] Andreas Zeller. Learning the language of failure. 2020 CASA Disinguished Lecture, 2020. URL: `https://andreas-zeller.info/assets/CASA-2020-Learning-the-Language-of-Failure.pdf`.

The Recursive Arrival Problem

Thomas Webster
University of Edinburgh, UK
Thomas.Webster@ed.ac.uk

We study an extension of the `Arrival` problem, called `Recursive Arrival`, inspired by Recursive State Machines, which allows for a family of switching graphs that can call each other in a recursive way. We study the computational complexity of deciding whether a Recursive Arrival instance terminates at a given target vertex. We show this problem is contained in $\mathsf{NP} \cap \mathsf{coNP}$, and we show that a search version of the problem lies in UEOPL, and hence in $\mathsf{EOPL} = \mathsf{PLS} \cap \mathsf{PPAD}$. Furthermore, we show P-hardness of the Recursive Arrival decision problem. By contrast, the current best-known hardness result for `Arrival` is PL-hardness.

1 Introduction

`Arrival` is a simply described decision problem defined by Dohrau, Gärtner, Kohler, Matoušek and Welzl [5]. Informally, it asks whether a train moving along the vertices of a given directed graph, with n vertices, will eventually reach a given target vertex, starting at a given start vertex. At each vertex, v, there are (without loss of generality) two outgoing edges and the train moves deterministically, alternately taking the first out-edge, then the second, switching between them if and when it revisits that vertex repeatedly. This process is known as "switching" and can be viewed as a deterministic simulation of a random walk on the directed graph. It can also be viewed as a natural model of a state transition system where a local deterministic cyclic scheduler is provided for repeated transitions out of each state.

Dohrau et al. showed that this `Arrival` decision problem lies in the complexity class $\mathsf{NP} \cap \mathsf{coNP}$, but it is not known to be in P. There has been much recent work showing that a search version of the `Arrival` problem lies in subclasses of TFNP including PLS [17], CLS [13], and UEOPL [12], as well as showing that `Arrival` is in $\mathsf{UP} \cap \mathsf{coUP}$ [13]. There has also been progress on lower bounds, including PL hardness and CC hardness [18]. Further, another recent result by Gärtner et al. [14] gives an algorithm for `Arrival` with running time $2^{\mathcal{O}(\sqrt{n}\log(n))}$, the first known sub-exponential algorithm. In addition, they give a polynomial-time algorithm for "almost acyclic" instances. Auger et al. also give a polynomial-time algorithm for instances on a "tree-like multigraph" [2].

The complexity of `Arrival` is particularly interesting in the context of other games on graphs. For instance, Condon's simple stochastic games, mean-payoff games, and parity games [4, 20, 16], where the two-player variants are known to be in $\mathsf{NP} \cap \mathsf{coNP}$ and the one-player variants have polynomial time algorithms. `Arrival`, however, is a zero-player game that has no known polynomial time algorithm and, furthermore, Fearnley et al. [11] that a one-player generalisation of Arrival is, in fact, NP-complete, in stark contrast to these two-player graph games.

We introduce and consider a new generalisation of Arrival that we call `Recursive Arrival`, in which we are given a finite collection of Arrival instances ("components") with the ability to, from certain nodes, invoke each other in a potentially recursive way. Each component has a set of entries and a set of exits, and we study the complexity of deciding whether the run starting from a given entry of a given component reaches a given exit of that component, which may involve recursive calls to other components.

A. Achilleos and D. Della Monica (Eds.): Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification (GandALF 2023).
EPTCS 390, 2023, pp. 168–184, doi:10.4204/EPTCS.390.11

Our model is inspired by work on recursive transition systems started by Alur et al. [1] with Recursive State Machines (RSMs) modelling sequential imperative programming. These inspired further work on Recursive Markov Chains (RMCs), Recursive Markov Decision Processes (RMDPs), and Recursive Simple Stochastic Games (RSSGs) by Etessami and Yannakakis [8, 9, 10]. RSMs (and RMCs) are essentially "equivalent" (see [9]) to (probabilistic) pushdown systems [3, 6] and have applications in model-checking of procedural programs with recursion.

There is previous work on Arrival generalisations including a variant we call `Succinct Arrival`, where at a vertex v the alternation takes the first outgoing edge of v on the first A_v visits and then the second edge the next B_v visits, repeating this sequence indefinitely. The numbers A_v and B_v are given succinctly in binary as input, and hence may be exponentially larger than the bit encoding size of the instance. Fearnley et al. showed that `Succinct Arrival` is P-hard and in $\mathsf{NP} \cap \mathsf{coNP}$ [11]. However, we do not know any inter-reducibility between `Recursive Arrival` and `Succinct Arrival` variants. In [19], we also defined and studied a generalisation of `Arrival` that allows both switching nodes as well as randomised nodes, and we showed that this results in PP-hardness and containment in PSPACE for (quantitative) reachability problems.

In this paper, we show that the `Recursive Arrival` problem lies in $\mathsf{NP} \cap \mathsf{coNP}$, like `Arrival`, by giving a generalised witness scheme that efficiently categorises both terminating and non-terminating instances. We also show that the natural search version of `Recursive Arrival` is in both PLS and PPAD and in fact in UEOPL, by giving a reduction to a canonical UEOPL problem. We also show P-hardness for the `Recursive Arrival` problem by reduction from the Circuit Value Problem. This contrasts with the current best-known hardness result for `Arrival`, which is PL-hardness ([18]).

Due to space limitations, many proofs are relegated to the full version of the paper.

2 Preliminaries

Let $\mathbb{N} = \{0, 1, \ldots\}$ denote the natural numbers, and let $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$. We assume the usual ordering on elements of $\mathbb{N}_\infty$. For $j \in \mathbb{N}$ and $k \in \mathbb{N}_\infty$, we use the notion $[j \ldots k] = \{i \in \mathbb{N} \mid j \leq i \leq k\}$, and we define $[k] = [1 \ldots k]$. All propositions of this section follow directly from the cited prior works.

Definition 2.1. A *switch graph* is given by a tuple $G := (V, s^0, s^1)$ where, for each $\sigma \in \{0,1\}$, $s^\sigma : V \to V$ is a function from vertices to vertices. $\square$

Given a Switch Graph G, we define its directed *edges* to be the set $E := \{(v, s^0(v)) \mid v \in V\} \cup \{(v, s^1(v)) \mid v \in V\}$, with self-loops allowed. We write $E_\sigma := \{(v, s^\sigma(v)) \mid v \in V\}$ for $\sigma \in \{0,1\}$ to refer to edges arising specifically from transitions $s^\sigma(v)$, for each vertex v.

Given a switch graph, $G := (V, s^0, s^1)$, we say $q : V \to \{0,1\}$ is a *switch position* on V. We let Q be the set of all switch positions on V and define $q^0 \in Q$ by $q^0(v) = 0$ for all $v \in V$ as the *initial switch position*. Given a switch graph, we say a *state* of the graph is an ordered pair $(v,q) \in V \times Q$ and we let $\Gamma = V \times Q$ be the *state space*. We define the "flip action", *flip* $: V \times Q \to Q$, of a vertex on a switch position, as follows: $\mathit{flip}(v,q)(u) = q(u)$ for all $u \in V \setminus \{v\}$ and $\mathit{flip}(v,q)(v) = 1 - q(v)$, i.e., this action flips the function value of q at v only. We can then define a transition function $\delta : \Gamma \to \Gamma$ on a switch graph as $\delta((v,q)) = (s^{q(v)}(v), \mathit{flip}(v,q))$.

We define the *run* of a switch graph G with initial state $\gamma_0 := (v_0, q_0)$ to be the unique infinite sequence over Γ, $\mathsf{RUN}^\infty(G, \gamma_0) := (\gamma_i)_{i=0}^\infty$ satisfying $\gamma_{i+1} := \delta(\gamma_i)$ for $i \geq 0$. For a vertex $v \in V$, we say a run *terminates at* v if $\exists t \in \mathbb{N}$ such that $\forall i \geq t\ \exists q_i \in Q$ with $\gamma_i = (v, q_i)$. We call $T \in \mathbb{N}_\infty$ the *termination time* defined by $T := \inf\{t \mid \forall i \geq t,\ v_i = v_t\}$, where $\inf \emptyset = \infty$. We denote by $\mathsf{RUN}(G, \gamma_0) := (\gamma_i)_{i=0}^T$ the subsequence of $\mathsf{RUN}^{\infty}(G,\gamma_0)$ up to the termination time T. We say a run *hits* a vertex $v \in V$ if $\exists t \in \mathbb{N}$ and $\exists q_t \in Q$ such that $\gamma_t = (v, q_t)$.

We note that in order to terminate at a vertex, $v \in V$, we must have that $v = s^0(v) = s^1(v)$. We define the set of "Dead Ends" in the instance as $\mathsf{DE} := \{v \in V \mid s^0(v) = s^1(v) = v\}$. From this definition, it is obvious that we either terminate at some unique vertex $v \in \mathsf{DE}$, or we never terminate. We may now define the Arrival Decision problem:

`Arrival`	
Instance:	A Switch Graph $G := (V, s^0, s^1)$ and vertices $o, d \in V$.
Problem:	Decide whether or not the run of switch graph G with initial state (o, q^0) terminates at vertex d.

Given a switch graph G with directed edges, E, we define the relations $\to^*, \to^+ \subseteq V \times V$ as follows $u \to^* v$ (resp. $\to^+$) for $u, v \in V$ if and only if there is a directed path $w_0, \ldots, w_k \in V$ with $(w_i, w_{i+1}) \in E$ for $i \in [k-1]$, with $w_0 = u$ and $w_k = v$ for $k \geq 0$ (resp. $k \geq 1$) from u to v in (V, E). We write $u \not\to^* v$ (resp. $\not\to^+$) whenever we do not have $u \to^* v$ (resp. $u \to^+ v$).

We note that we can view the sequence of vertices visited on a run as a directed path in (V, E), thus if the run with initial state (v, q) hits w then we can conclude $v \to^* w$ and, contrapositively, if $v \not\to^* w$ then for all $(v, q) \in Q$ the run starting at (v, q) does not hit w.

We let $\mathbb{I}\{a = b\}$ be the indicator function of $a = b$, which is equal to 1 if $a = b$ and is equal to 0 otherwise. We now define a switching flow, rephrasing Definition 2 of Dohrau et al. [5]:

Definition 2.2 ([5, Definition 2])**.** Let $G := (V, s^0, s^1)$ be a switch graph, and let $o, d \in V$ be vertices. We define a *switching flow* on G from o to d as a vector $\boldsymbol{x} := (x_e \mid e \in E)$ where $x_e \in \mathbb{N}$ such that the following family of conditions hold for each $v \in V$:

$$\text{Flow Conservation}: \left(\sum_{e=(u,v)\in E} x_e\right) - \left(\sum_{e=(v,w)\in E} x_e\right) = \mathbb{I}\{v = d\} - \mathbb{I}\{v = o\}, \quad \forall v \in V,$$

$$\text{Parity Condition}: \quad x_{(v,s^1(v))} \leq x_{(v,s^0(v))} \leq x_{(v,s^1(v))} + 1, \qquad \forall v \in V. \qquad \square$$

We note that given G, o and a switching flow $\boldsymbol{x}$ from o to some, unknown, vertex $d \in V$, we can compute exactly which d by verifying the equalities. We refer to d as the *current-vertex* of the switching flow. If $o \in V$ is an initial vertex and $t \in \mathbb{N}$ a time, we let $\mathsf{RUN}(G, (o, q^0)) := ((v_i, q_i))_{i=0}^{\infty}$ be the run, and define the *Run Profile* to time t to be the vector $\boldsymbol{run}(o,t) := (|\{i \in [t] \mid (v_{i-1}, v_i) = e\}| \mid e \in E)$. It follows that for any $o \in V$ and $t \in \mathbb{N}$ that $\boldsymbol{run}(o,t)$ is a switching flow from o to some vertex $d \in V$ [5, Observation 1]. We say a switching flow $\boldsymbol{x}$ is *run-like* if there exists some $t \in \mathbb{N}$ such that $\boldsymbol{x} = \boldsymbol{run}(o,t)$.

It follows directly from the results of Dohrau et al.[5] and Gartner et al.[13] that:

Proposition 2.3 ([5, 13])**.** *There exists a polynomial function* $\mathrm{p} : \mathbb{N} \to \mathbb{N}$ *such that for all Switch Graphs* $G := (V, s^0, s^1)$ *and all vertices* $o, d \in V$ *with* $o \neq d$ *and* $d \in \mathsf{DE}$ *the following are equivalent:*

- *The run on G from initial state (o, q^0) terminates at d.*
- *There exists a run-like switching flow $\boldsymbol{x}$ on G from o to d satisfying $\forall e \in E$, that* $\log_2(x_e) \leq \mathrm{p}(|G|)$.

Furthermore, for the same polynomial p*, the following are equivalent:*

- *The run on G from initial state (o, q^0) does not terminate.*
- *There exists a vertex $d' \in V \setminus \mathsf{DE}$, a run-like switching flow $\boldsymbol{x}$ on G from o to d', and an edge $e' = (u, d') \in E$ which satisfies for all $e \in E \setminus \{e'\}$ that* $\log_2(x_e) \leq \mathrm{p}(|G|)$ *and that* $x_{e'} = 2^{\mathrm{p}(|G|)} + 1$.

It follows from these results that `Arrival` is in $\mathsf{NP} \cap \mathsf{coNP}$, as we may non-deterministically guess a vector, $\boldsymbol{x}$, whose coordinate entries are bounded by $2^{\mathsf{p}(|G|)} + 1$, and then verify whether or not $\boldsymbol{x}$ is a run-like switching flow. Using [13, Lemma 11] we may verify the run-like condition in polynomial time, on which we will elaborate subsequently. If we find a run-like switching flow to some dead end $d' \in \mathsf{DE}$ we may conclude G terminates at d' and by the first part of Proposition 2.3 we can find such a flow within these bounds. This may be either a flow to the given dead-end d in our input, or to some other dead-end, certifying non-termination at d. The last case of Proposition 2.3 says that when G does not terminate anywhere, we may also find a flow certifying this within our bounds, namely with some coordinate value of the guessed vector $\boldsymbol{x}$ being exactly$2^{\mathsf{p}(|G|)} + 1$. In fact, it was shown by [13] that this argument also shows containment of `Arrival` in $\mathsf{UP} \cap \mathsf{coUP}$, by showing there is a unique witness $\boldsymbol{x}$ satisfying just one of these conditions.

Let $G := (V, s^0, s^1)$ be a Switch Graph and let $\boldsymbol{x}$ be a switching flow on G between some vertices $o, d \in V$. We define the *last-used-edge* graph $G^*_{\boldsymbol{x}} := (V, E^*_{\boldsymbol{x}})$ with the following set of edges:

$$E^*_{\boldsymbol{x}} := \{(v, s^0(v)) \mid v \in V \text{ and } x_{(v,s^0(v))} \neq x_{(v,s^1(v))}\} \cup \{(v, s^1(v)) \mid v \in V \text{ and } x_{(v,s^0(v))} = x_{(v,s^1(v))} > 0\}$$

This graph contains at most one of the edges $(v, s^0(v))$ or $(v, s^1(v))$. If $x_{(v,s^0(v))} + x_{(v,s^1(v))} > 0$, then assuming there exists some run on which we visit vertex v a total of $x_{(v,s^0(v))} + x_{(v,s^1(v))}$ times, $E^*_{\boldsymbol{x}}$ contains the edge out of v that our switching order would use the last time v was visited on such a run. If on the other hand $x_{(v,s^0(v))} + x_{(v,s^1(v))} = 0$, then $E^*_{\boldsymbol{x}}$ contains neither edge.

Proposition 2.4 ([13])**.** *Let $G := (V, s^0, s^1)$ be a Switch Graph and let $\boldsymbol{x}$ be a switching flow on G from $o \in V$ to $d \in V$, then there exists a unique $t \in \mathbb{N}$ such that $\boldsymbol{x} = \boldsymbol{run}(o,t)$, if and only if one of the following two (mutually exclusive) conditions hold:*

- *The graph $G^*_{\boldsymbol{x}}$ is acyclic,*
- *The graph $G^*_{\boldsymbol{x}}$ contains exactly one cycle and d is on this cycle,*

Furthermore, given G and any such $\boldsymbol{x}$ whether or not one of these conditions hold can be checked in polynomial time in the size of G and $\boldsymbol{x}$.

Proposition 2.5 ([13, Lemma 16])**.** *Let $G := (V, s^0, s^1)$ be a Switch Graph and let $t \in \mathbb{N}$ with $\boldsymbol{run}(o,t)$ the run profile up to time t, which is a switching flow on G from $o \in V$ to some vertex $d \in V$. Then at least one of the following two conditions hold:*

- *There is a unique edge $(u,d) \in E^*_{\boldsymbol{run}(o,t)}$ incoming to d in the graph $G^*_{\boldsymbol{run}(o,t)}$.*
- *The graph $G^*_{\boldsymbol{run}(o,t)}$ contains exactly one cycle, and that cycle contains exactly one edge of the form $(u,d) \in E^*_{\boldsymbol{run}(o,t)}$ on the cycle.*

Moreover, the edge (u,d) was the edge traversed at time t in the run (i.e., if $\mathsf{RUN}^\infty(G,(o,q^0)) = ((v_i,q_i))_{i=0}^\infty$ then $v_{t-1} = u$ and $v_t = d$). Furthermore, this uniquely determined edge can be computed given G and $\boldsymbol{run}(o,t)$ in time polynomial in the size of G and $\boldsymbol{run}(o,t)$.

Using these results, we are able to efficiently (in P-time) compute a function LUE which takes a switching flow of the form $\boldsymbol{run}(o,t)$ and returns the "last-used-edge", namely the unique edge $(u,d) \in E$ guaranteed by Proposition 2.5, where (u,d) is the edge which was traversed at time t.

2.1 The Recursive Arrival Problem

We consider a recursive generalisation of Arrival in the spirit of Recursive State Machines, etc. ([1, 9, 10]). A Recursive Arrival instance is defined as follows:

Definition 2.6. A *Recursive Arrival* graph is given by a tuple, $(G^1, \dots, G^k)$, where each *component* $G^i := (N_i \cup B_i, Y_i, \mathsf{En}_i, \mathsf{Ex}_i, \delta_i)$ consists of the following pieces:

- A set N_i of *nodes* and a (disjoint) set B_i of *boxes*.
- A labelling $Y_i : B_i \to \{1, \dots, k\}$ that assigns every box an index of one of the components $G^1, \dots, G^k$.
- A set of *entry nodes* $\mathsf{En}_i \subseteq N_i$ and a set of *exit nodes* $\mathsf{Ex}_i \subseteq N_i$.
- To each box $b \in B_i$, for all $i \in [k]$, we associate a set of *call ports*, $\mathsf{Call}_{b,i} = \{(b,o) \mid o \in \mathsf{En}_{Y_i(b)}\}$ corresponding to the entries of the corresponding component, and a set of *return ports*, $\mathsf{Return}_{b,i} = \{(b,d) \mid d \in Ex_{Y_i(b)}\}$ corresponding to the exits of the corresponding component. We define the sets $\mathsf{Call}_i = \cup_{b \in B_i} \mathsf{Call}_{b,i}$ and $\mathsf{Return}_i = \cup_{b \in B_i} \mathsf{Return}_{b,i}$. We will use the term *ports* of G^i to refer to the set $\mathsf{Port}_i = \mathsf{Call}_i \cup \mathsf{Return}_i$, of all call ports and return ports associated with all boxes $b \in B_i$ that occur within the component G^i.
- A *transition relation*, δ_i, where transitions are of the form (u, σ, v) where:
  1. The source u is either a node in $N_i \setminus \mathsf{Ex}_i$ or a return port (b,x) in Return_i. We define $\mathsf{Sor}_i = N_i \setminus \mathsf{Ex}_i \cup \mathsf{Return}_i$ to be the set of all *source vertices*.
  2. The label σ is either 0 or 1.
  3. The destination v is either a node in $N_i \setminus \mathsf{En}_i$ or a call port (b,e) where b is a box in B_i and e is an entry node in En_j for $j = Y_i(b)$; we call this the set Dest_i of *destination vertices*.

  and we require that the relation δ_i has the following properties:

  1. For every vertex $u \in \mathsf{Sor}_i$ and each $\sigma \in \{0,1\}$ there is a unique vertex $v \in \mathsf{Dest}_i$ with $(u, \sigma, v) \in \delta_i$. Thus, for each $i \in [k]$ and $\sigma \in \{0,1\}$, we can define total functions $s_i^\sigma : \mathsf{Sor}_i \to \mathsf{Dest}_i$ by the property that $(u, \sigma, s_i^\sigma(u)) \in \delta_i$, for all $u \in \mathsf{Sor}_i$. □

We will use the term *vertices* of G^i, which we denote by V_i to refer to the union $V_i = N_i \cup \mathsf{Port}_i$ of its set of nodes and its set of ports. For $\sigma \in \{0,1\}$, we let $E_i^\sigma = \{(u,v) \mid (u,\sigma,v) \in \delta_i\}$ be the set of underlying edges of δ_i with label σ, and we define $E_i := E_i^0 \cup E_i^1$. We will often alternatively view components as being equivalently specified by the pair of functions (s_i^0, s_i^1), which define the transition function $\delta_i := \{(u, \sigma, s_i^\sigma(u)) \mid u \in \mathsf{Sor}_i, \sigma \in \{0,1\}\}$.

We can view a box as a "call" to other components, and, as such, it is natural to ask which components "call" other components. Given an instance of Recursive Arrival, $(G^1, \dots, G^k)$, we define its *Call Graph* to be the following directed graph, $C = ([k], E_C)$. Our vertices are component indices and for all $(i, j) \in [k] \times [k]$ let $(i, j) \in E_C$ if and only if there exists some $b \in B_i$ with $j = Y_i(b)$ (i.e., a component G^i can make a call to component G^j). We allow self-loop edges in this directed graph, which correspond to a component making a call to itself.

We are also able to lift some definitions from non-recursive Arrival to analogous definitions about Recursive Arrival instances. Firstly, we define the sets $\mathsf{DE}_i := \{v \in \mathsf{Sor}_i \mid s_i^0(v) = s_i^1(v) = v\} \cup \mathsf{Ex}_i$, of dead-ends of each component. This contains both vertices $v \in \mathsf{Sor}_i$ where both outgoing transitions are to itself and all the exits of the component.

In a given component, G^i, we define a *switch position* on G^i as a function $q : \mathsf{Sor}_i \to \{0,1\}$. We let Q_i be the set of all switch position functions on G^i. We let $q_i^0 \in Q_i$ be the function $q_i^0(v) = 0$ for all

$v \in \mathsf{Sor}_i$ and call this the *initial switch position*. We define the action *flip*$_i : \mathsf{Sor}_i \times Q_i \to Q_i$ analogously to non-recursive Arrival, which flips the bit corresponding to a given vertex in a given switch position.

A *state* of a Recursive Arrival graph $(G^1, \dots, G^k)$ is given by a tuple $\gamma := ((b_1, q_1) \dots (b_r, q_r), (v, q))$ where the *call stack* $\beta := (b_1, q_1) \dots (b_r, q_r)$ is a string of pairs (b_i, q_i) with each $b_i \in \cup_k B_k$ a box, q_i is a switch position on some component G^{c_i} (i.e. $q_i \in Q_{c_i}$), and the *current position* is the pair (v, q) where $v \in V_{c_{r+1}}$ is a vertex in some component $G^{c_{r+1}}$ and $q \in Q_{c_{r+1}}$ is a switch position on $G^{c_{r+1}}$. We call the sequence $(c_1, \dots, c_r, c_{r+1})$ the *component call-stack* of the state. We say that a state is *well-formed* if:

- For all $i \in [r]$ we have $b_i \in B_{c_i}$.
- The sequence satisfies $Y_{c_i}(b_i) = c_{i+1}$ for $i \in [r]$.

We let Γ be the set of all well-formed states and $\Gamma_S := \{\beta : \exists (v, q),\ (\beta, (v, q)) \in \Gamma\}$ be the set of well-formed stacks β appearing in some state of Γ.

We define the transition function $\delta : \Gamma \to \Gamma$ on a well-formed state $\gamma := ((b_1, q_1) \dots (b_r, q_r), (v, q))$ as:

1. If $v \in \mathsf{Sor}_j$ is a source vertex then we let $v' := s^{q(v)}(v)$ and then we define $\delta(\gamma) := ((b_1, q_1), \dots, (b_r, q_r), (v', \mathit{flip}_j(v, q)))$;
2. If $v = (b, e) \in \mathsf{Call}_j$ then $e \in \mathsf{En}_{j'}$ for $j' = Y_j(b)$. We let $q^0_{j'}$ be the initial switch position on $G^{j'}$ and define $\delta(\gamma) := ((b_1, q_1) \dots (b_r, q_r)(b, q), (e, q^0_{j'}))$;
3. If $v \in \mathsf{Ex}_j$ and $r \geq 1$ then we define $\delta(\gamma) := ((b_1, q_1) \dots (b_{r-1}, q_{r-1}), ((b_r, v), q_r))$;
4. If $v \in \mathsf{Ex}_j$ and $r = 0$ then $\delta(\gamma) := \gamma$;

The function $\delta : \Gamma \to \Gamma$ defines a deterministic transition system on well-formed states. We call the *run* of a Recursive Arrival graph from an initial component index $j \in [k]$, an initial switch position $q_0 \in Q_j$ and a start entrance $o \in \mathsf{En}_j$ the (infinite) sequence $\mathsf{RUN}^\infty(G, (o, q_0)) := (\gamma_i)_{i=0}^\infty$ given by $\gamma_0 := (\varepsilon, (o, q_0))$ and $\gamma_{i+1} := \delta(\gamma_i)$. We say a run *terminates* at an exit $d \in \mathsf{Ex}_j$ if there $\exists t \in \mathbb{N}$ such that $\forall i \geq t$ there $\exists q_i \in Q_j$ such that $\gamma_i = (\varepsilon, (d, q_i))$. We call $T \in \mathbb{N}_\infty$ the termination time defined by $T := \inf\{t \mid \forall i \geq t,\ v_i \in \mathsf{Ex}_j\}$, where $\inf(\emptyset) = \infty$. We denote by $\mathsf{RUN}(G, (o, q)) := (\gamma_i)_{i=0}^T$ the subsequence up to termination. We say a run *hits* a vertex $v \in V$ if there $\exists t \in \mathbb{N}$, $\exists q_t \in Q$ and $\exists \beta \in \Gamma_S$ with $\gamma_t = (\beta, (v, q_t))$.

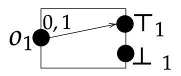
Our decision problem can then be stated as:

`Recursive Arrival`

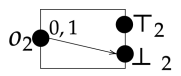
Instance: A Recursive Arrival graph $(G^1, \dots, G^k)$, with $|\mathsf{En}_j| = 1$ for all $j \in [k]$, and a target exit $d \in \mathsf{Ex}_1$

Problem: Does the run from initial state $(\varepsilon, (o_1, q^0_1))$ terminate at exit d? (Where $o_1 \in \mathsf{En}_1$ is the unique entry of G^1 and $q^0_1 \in Q_1$ is the initial switch position.)

This decision problem covers in full generality any termination decision problem on Recursive Arrival instances, as we may accomplish a change of initial state by renumbering components and relabelling transitions. Also, restricting to models with $|\mathsf{En}_i| = 1$ is without loss of generality, because we can efficiently convert the model into an “equivalent” one where each component has a single entry, by making copies of components (and boxes) with multiple entries, each copy associated with a single entry (single, call port, respectively). This is analogous to the same fact for Recursive Markov Chains, which was noted by Etessami and Yannakakis in [9, p. 16]. Thus, we may assume that in the `Recursive Arrival` problem all components of the instance have a unique entry, i.e., for $i \in [k]$ that $\mathsf{En}_i = \{o_i\}$, and, unless stated otherwise, the run on G refers to the run starting in the state $(\varepsilon, (o_1, q^0_1))$, writing $\mathsf{RUN}(G) := \mathsf{RUN}(G, (o_1, q^0_1))$.

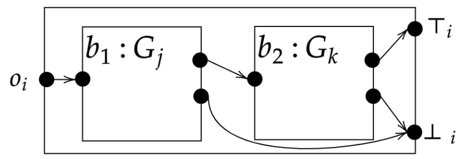


(a) G^1, the constant “true” component.

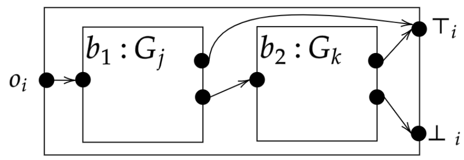


(b) G^2, the constant “false” component.

Figure 1: Initial components corresponding to constant gates “true" and “false".



(a) Component for the AND of gates g_j and g_k.



(b) Component for the OR of gates g_j and g_k.

Figure 2: Component G^i, where $j, j' \in [i-1]$ are the indices of the two inputs to the gate g_i. All edges correspond to both s^0 and s^1 transitions.

While, in such an instance, we may make an exponential number of calls to other functions, it turns out we are able to give a polynomial bound on the maximum recursion depth before we can conclude an instance must loop infinitely.

Lemma 2.7. *Let $G := (G^1, \dots, G^k)$ be an instance of Recursive Arrival and assume the run on G hits some state $(\beta, (v, q))$, with $|\beta| \geq k$. Then the run on G does not terminate.*

3 P-Hardness of Recursive Arrival

Manuell [18] has shown the `Arrival` problem to be PL-hard, which trivially provides the same hardness result for `Recursive Arrival`. This is currently the strongest hardness result known for the `Arrival` problem. By contrast, we now show that the `Recursive Arrival` problem is in fact P-hard.

Theorem 3.1. *The* `2-exit Recursive Arrival` *problem is* P*-hard.*

Proof (Sketch). We show this by reduction from the P-complete Monotone Circuit Value Problem (see e.g., [15]). We construct one component corresponding to each gate of an input boolean circuit. Each component will have two exits, which we refer to as “top”, $\top$, and “bottom”, $\perp$, (located accordingly in our figures) and we will view these exits as encoding the outputs, “true” and “false” respectively.

Firstly, we show in Figure 1 two components for a constant true and constant false gate of the circuit. Depicted in Figure 2 are two cases corresponding to AND or OR gates. These perform a lazy evaluation of the AND or OR of components G^j and G^k. This process produces a polynomially sized `Recursive Arrival` instance for an input boolean circuit where each component G_j can be shown inductively to reach exit $\top_j$ if and only if it’s corresponding gate, g_j, outputs true. □

4 Recursive Arrival is in NP ∩ coNP and UEOPL

Recall the notion of Switching Flow for an Arrival instance. For Recursive Arrival, we generalise the notion of a Switching Flow to a tuple of vectors $(\boldsymbol{x}^1, \dots, \boldsymbol{x}^k)$, one for each component of the Recursive Arrival instance. We define for each component G^i, $i \in [k]$, and each box $b \in B_i$ the set of *potential edges* $F_{b,i} := \mathsf{Call}_{b,i} \times \mathsf{Return}_{b,i}$, representing the potential ways of crossing the box b, assuming that the box is eventually returned from. We define the sets $F_i := \cup_{b \in B_i} F_{b,i}$. We recall that the set of internal edges of a component G^i is given by $E_i := \{(u,v) \mid u,v \in V_i,\ \exists \sigma \in \{0,1\}, (u,\sigma,v) \in \delta_i\}$. We say the *Flow Space* for component G^i is the set of vectors $\mathscr{F}_i := \mathbb{N}^{|E_i \cup F_i|} := \{(x^i_e \in \mathbb{N} \mid e \in E_i \cup F_i)\}$, where we identify coordinates of these vectors with edges in $E_i \cup F_i$. We define the *Flow Space* of G to be the set $\mathscr{F} := \Pi_{i=1}^k \mathscr{F}_i$, a tuple of k vectors, with the i'th vector in the flow space of component G^i. We denote specifically by $\mathbf{0}^i \in \mathscr{F}_i$ the all zero vector, which has $\mathbf{0}^i_e = 0$ for all $e \in E_i \cup F_i$, and $\mathbf{0} \in \mathscr{F}$ the all zero tuple, $\mathbf{0} := (\mathbf{0}^1, \dots, \mathbf{0}^k)$. We refer to elements of $\mathscr{F}$ (resp. $\mathscr{F}_i$) as flows on G (resp. G^i).

Firstly, we define a switching flow on each component. For a Recursive Arrival instance $G := (G^1, \dots, G^k)$ and for $l \in [k]$, we call a vector $\boldsymbol{x}^l \in \mathscr{F}_l$ to be a *component switching flow* if the following conditions hold. Firstly, by definition, the all-zero vector $\mathbf{0}^l$ is always considered a component switching flow. Furthermore, by definition, a non-zero vector $\boldsymbol{x}^l \in \mathscr{F}_l \setminus \{\mathbf{0}^l\}$ is called a component switching flow if there exists some *current-vertex* $d^l_{\boldsymbol{x}^l} \in V_l \setminus \{o_l\}$ (which, as we will see, is always uniquely determined when it exists), such that for o_l the unique entry of G^l, $\boldsymbol{x}^l$ satisfies the following family of conditions:

$$\text{Flow Conservation} \quad \begin{cases} \left(\sum_{e=(u,v)\in E_l\cup F_l} x^l_e\right) - \left(\sum_{e=(v,w)\in E_l\cup F_l} x^l_e\right) = 1, & \text{for } v = d^l_{\boldsymbol{x}^l}, \\ \qquad\qquad\qquad\qquad + \left(\sum_{e=(v,w)\in E_l\cup F_l} x^l_e\right) = 1, & \text{for } v = o_l, \\ \left(\sum_{e=(u,v)\in E_l\cup F_l} x^l_e\right) - \left(\sum_{e=(v,w)\in E_l\cup F_l} x^l_e\right) = 0, & \forall v \in V_l \setminus \{o_l, d^l_{\boldsymbol{x}^l}\}, \end{cases}$$

$$\text{Switching Parity Condition} \quad x_{(v,s^1(v))} \le x_{(v,s^0(v))} \le x_{(v,s^1(v))} + 1, \qquad \forall v \in \mathsf{Sor}_l,$$

$$\text{Box Condition} \quad \exists f_b \in F_{b,l} \text{ such that } \forall f \in (F_{b,l} \setminus \{f_b\})\ x^l_f = 0, \qquad \forall b \in B_l$$

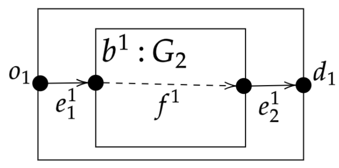
Importantly, note that for any such component switching flow, $\boldsymbol{x}^l$, the current-vertex node $d^l_{x^l}$ is *uniquely* determined. This follows from the fact that the left-hand sides of the Flow Conservation equalities for nodes $v \in V_l \setminus \{o_l\}$ are identical and independent of the specific node v. Hence, if a vector $\boldsymbol{x}^l$ satisfies all of those equalities, there can only be one vertex $v \in V_l \setminus \{o_l\}$ for which the corresponding linear expression on the left-hand side, evaluated over the coordinates of the vector $\boldsymbol{x}^l$, equals 1.

In the case where $\boldsymbol{x}^l = \mathbf{0}^l$, i.e., the all zero-vector, we define the current-vertex of the all-zero component switching flow to be $d^l_{\mathbf{0}^l} := o_l$. We say a component switching flow $\boldsymbol{x}^l \in \mathscr{F}_l$ is *complete* if its current vertex $d^l_{\boldsymbol{x}^l}$ is an exit vertex in Ex_l. These conditions follow the same structure as for non-recursive switching flows, with the additional "Box Condition" only allowing at most one potential edge across each box (i.e., an edge in $F_{b,l}$) to be used.

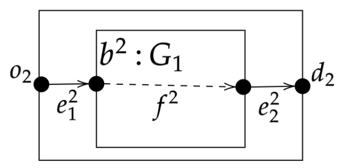
Next, we extend our component switching flows by adding conditions that relate the flows on different components. Consider a tuple $\boldsymbol{X} := (\boldsymbol{x}^1, \dots, \boldsymbol{x}^k) \in \mathscr{F}$ of vectors, one for each component, such that each $\boldsymbol{x}^i \in \mathscr{F}_i$ is a component switching flow for component G^i. We sometimes write $d^i_{\boldsymbol{X}}$ instead of $d^i_{\boldsymbol{x}^i}$. Let $K_{\boldsymbol{X}} = \{i \in [k] \mid \boldsymbol{x}^i \text{ is complete}\}$ be the subset of indices corresponding to complete component switching flows. We then say the tuple $\boldsymbol{X} \in \mathscr{F}$ is a *recursive switching flow* if for every $l \in [k]$, $b \in B_l$ and $f \in F_{b,l}$, the following holds:

- $\boldsymbol{x}^l \in \mathscr{F}_l$ is a component switching flow for component G^l, and
- if $x^l_f > 0$ then $Y_l(b) \in K_{\boldsymbol{X}}$, and
- if $x^l_f > 0$, then letting $d^{Y_l(b)}_{\boldsymbol{X}} \in \mathsf{Ex}_{Y_l(b)}$ be the current vertex of $\boldsymbol{x}^{Y_l(b)}$, we must have that $f = ((b, o_{Y_l(b)}), (b, d^{Y_l(b)}_{\boldsymbol{X}}))$.

We define $\mathscr{R} \subset \mathscr{F}$ to be the set of all recursive switching flows. These conditions ensure "consistency" in the following way; if we use an edge $f \in F_{b,l}$ then we have a component switching flow on component



(a) Initial component G^1.



(b) Component G^2.

Figure 3: A Recursive Arrival instance G on which there exists a recursive switching flow $(\boldsymbol{x}^1,\boldsymbol{x}^2)$ on G whose current vertex is in G^1 is d_1 however the run on G does not terminate, or even hit the exit d_1.

$G^{Y_l(b)}$ which is complete and reaches the exit matching the edge f, and we are taking that same edge across all boxes with the same label. We note our definition implies $\mathbf{0} \in \mathscr{R}$, thus there is always at least one recursive switching flow. These conditions can be verified in polynomial time.

We will view recursive switching flows as hypothetical partial "runs" on each component, where an edge $e \in E_l \cup F_l$ is used x_e^l times along this "run". It may well be the case no such run actually exists. However, unlike the case of non-recursive switching flows in Arrival, it is no longer the case that any recursive switching flow where the current vertex is $d_{\boldsymbol{X}}^1$ in component G^1, and where $d_{\boldsymbol{X}}^1 \in \mathsf{Ex}_1$ is an exit, necessarily certifies termination at $d_{\boldsymbol{X}}^1$. It need not do so. For example, in the instance depicted in Figure 3 we may give the following flow on G: $\boldsymbol{x}^1 = (1,1,1)$, $\boldsymbol{x}^2 = (1,1,1)$. The instance depicted obviously loops infinitely, alternating calls between components G^1 and G^2, but neither ever reaching an exit. However, the given $(\boldsymbol{x}^1,\boldsymbol{x}^2)$ corresponds to a recursive switching flow for this instance, both of whose component switching flows have an exit as their current vertex.

We need a way to determine whether the recursive switching flow avoids such pathologies. To do this, we need some additional definitions. We describe a component switch flow $\boldsymbol{x}^l$ as *call-pending* if its current vertex $d_{\boldsymbol{x}^l}^l \in \mathsf{Call}_l$ is a call port, we let $J_{\boldsymbol{X}} \subseteq [k]$ be the set of all call-pending components and we let $r_{\boldsymbol{X}} := |J|$. From a recursive switching flow $\boldsymbol{X} := (\boldsymbol{x}^1,\ldots,\boldsymbol{x}^k)$ we can compute the *pending-call graph* $C_{\boldsymbol{X}}^{\mathrm{Pen}} := ([k], E_{\boldsymbol{X}}^{\mathrm{Pen}})$ where we have edge $(i,j) \in E_{\boldsymbol{X}}^{\mathrm{Pen}}$ if and only if $i \in J_{\boldsymbol{X}}$, $d_{\boldsymbol{X}}^i = (b,o) \in \mathsf{Call}_i$ is the current vertex of $\boldsymbol{x}^i$ and $j = Y_i(b)$. We can also compute the *completed-call graph*, $C_{\boldsymbol{X}}^{\mathrm{Com}} := ([k], E_{\boldsymbol{X}}^{\mathrm{Com}})$, where we have an edge $(i,j) \in E_{\boldsymbol{X}}^{\mathrm{Com}}$ if and only if $\exists b \in B_i$, $\exists f \in F_{i,b}$ with $x_f^i > 0$ and $Y_i(b) = j$. The pending-call graph represents, from the perspective of an imagined "run" corresponding to the recursive switching flow $\boldsymbol{X}$, which components G^i are currently "paused" at a call port and waiting for component G^j to reach an exit to determine the return port they should move to next. The completed-call graph represents the dependencies in the calls already made in such an imagined run, where an edge from component G^i to component G^j means that inside component G^i the imagined run is making a call to a box labelled by G^j and "using" the fact that component G^j, once called upon, reaches a specific exit. In turn, in order to G^j to reach its exit the imagined run might be "using" the completion of other components to which there are outgoing edges from G^j in the completed-call graph. Thus, any cycle in the completed-call graph represents a series of circular (and hence not well-founded) assumptions about the imagined "run" corresponding to the recursive switching flow $\boldsymbol{X}$. For example, in the case of a 2-cycle between components G^i and G^j, these are: "If G^i reaches exit $d_{\boldsymbol{X}}^i$ then G^j reaches exit $d_{\boldsymbol{X}}^j$"; and "If G^j reaches exit $d_{\boldsymbol{X}}^j$ then G^i reaches exit $d_{\boldsymbol{X}}^i$" (c.f. Figure 3).

Let G be an instance of recursive arrival and let $\mathsf{RUN}^\infty(G, o_1, q_1^0) := (\beta_t, (v_t, q_t))_{t=0}^\infty$ be the run starting at (o_1, q_1^0). We define the times $S_l := \inf\{t \mid v_t = o_l\}$ and $T_l := \inf\{t \mid v_t \in \mathsf{Ex}_l\}$ for each component index $l \in [k]$, with these values being ∞ if the set is empty. If $S_l < \infty$ we define the stack $\beta^l := \beta_{S_l}$. We define the *component run* to be the (potentially finite) subsequence $t_1^l, t_2^l, \ldots$ of times which are precisely all times

$t^l_j \in [S_l, \ldots, T_l]$ where $\beta_{t^j_l} = \beta^l$. We define the *Recursive Run Profile* of G up to time t as the sequence of vectors, $\boldsymbol{Run}(G,t) := (\boldsymbol{run}(G^1,t), \ldots, \boldsymbol{run}(G^k,t))$, where for each $l \in [k]$, $\boldsymbol{run}(G^l,t) := (|\{j \in \mathbb{N} \mid t^l_{j+1} \leq t \wedge (v_{t^l_j}, v_{t^l_{j+1}}) = e\}| \mid e \in E_l \cup F_l)$.

In other words, $\boldsymbol{run}(G^l,t)$ is a vector that provides counts of how many times each edge in component G^l has been crossed, up to time t, during one "visit" to component G^l, with some particular call stack. (The specific call stack doesn't matter. This sequence does not depend on the specific calling context β_l in which G^l was initially called.) We note that $\boldsymbol{run}(G^l,0) = \mathbf{0}^l$.

Similarly to the non-recursive case, we can define the *last-used-edge* graph for each component G^l as, $G^*_{l,\boldsymbol{x}^l} := (V_l, E^*_{l,\boldsymbol{x}^l})$ who's edge set is defined as:

$$\begin{aligned}E^*_{l,\boldsymbol{x}^l} := &\{(v,s^0(v)) \mid v \in \mathsf{Sor}_l \text{ and } x^l_{(v,s^0(v))} \neq x^l_{(v,s^1(v))}\} \cup \\ &\{(v,s^1(v)) \mid v \in \mathsf{Sor}_l \text{ and } x^l_{(v,s^0(v))} = x^l_{(v,s^1(v))} > 0\} \cup \{f \in F_l \mid x^l_f > 0\}\end{aligned}$$

We note that for the all-zero vector we have $E^*_{l,\mathbf{0}^l} = \emptyset$, and if $\boldsymbol{x}^l \neq \mathbf{0}^l$ is non-zero then the current vertex $d^l_{\boldsymbol{x}^l}$ must have at least one incoming edge in $E^*_{l,\boldsymbol{x}^l}$, and thus the set $E^*_{l,\boldsymbol{x}^l}$ isn't empty.

Depending on how our run evolves, there are three possible cases:

- For all $l \in [k]$, if $S_l < \infty$ then $T_l < \infty$. This case corresponds to reaching some exit of G^1, i.e., terminating there.
- There exists some $l \in [k]$ with $S_l < \infty$ and yet with $T_l = \infty$, however, where for all such $l \in [k]$ the subsequence $t^l_1, t^l_2, \ldots$ is of finite length. This case corresponds to blowing up the call stack to arbitrarily large sizes, and as we shall describe, we can detect it by looking for a cycle in $C^{\mathrm{Pen}}_{\boldsymbol{X}}$.
- There exists $l \in [k]$ with $S_l < \infty$ and $T_l = \infty$, where the subsequence $t^l_1, t^l_2, \ldots$ is of infinite length. This case corresponds to getting stuck inside component G^l, and infinitely often revisiting a vertex in a loop with the same call stack. As we shall see, we can detect this case by looking for a sufficiently large entry in some coordinate of $\boldsymbol{x}^l$.

Let G be a Recursive Arrival instance and let $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{R}$ be a recursive switching flow on G, we say $\boldsymbol{X}$ is *run-like* if it satisfies the following conditions:

- For each component index $l \in [k]$ one of the following two conditions hold:
  - The graph $G^*_{l,\boldsymbol{x}^l}$ is acyclic,
  - The graph $G^*_{l,\boldsymbol{x}^l}$ contains exactly one cycle and $d^l_{\boldsymbol{x}^l}$ is on this cycle.
- If the set of call-pending component indexes $J_{\boldsymbol{X}}$ is non-empty, then $1 \in J_{\boldsymbol{X}}$ and there is some total ordering $j_1, \ldots, j_{r_{\boldsymbol{X}}}$ of the set $J_{\boldsymbol{X}}$, with $j_1 = 1$, and a unique $j_{(r_{\boldsymbol{X}}+1)} \in [k]$ such that the edges of the pending-call graph are given by $E^{\mathrm{Pen}}_{\boldsymbol{X}} = \{(j_i, j_{i+1}) \mid i \in [r_{\boldsymbol{X}}]\}$. Note that we may have $j_{(r_{\boldsymbol{X}}+1)} = j_m$ for some $m \in \{1, \ldots, r_{\boldsymbol{X}}\}$, in which case $E^{\mathrm{Pen}}_{\boldsymbol{X}}$ forms not a directed line graph but a "lasso" meaning a directed line ending in one directed cycle. When $J_{\boldsymbol{X}} = \emptyset$ we say that $r_{\boldsymbol{x}} := 0$ and that $j_1 := 1$, thus the sequence is defined for all $\boldsymbol{X}$.
- For any $l \in [k]$ either: $l \in J_{\boldsymbol{X}} \cup K_{\boldsymbol{X}}$, or $\boldsymbol{x}^l = (0, \ldots, 0)$, or $l = j_{(r_{\boldsymbol{X}}+1)}$.
- The completed-call graph $C^{\mathrm{Com}}_{\boldsymbol{X}} := ([k], E^{\mathrm{Com}}_{\boldsymbol{X}})$ is acyclic.
- For any $l \in [k]$, if $\boldsymbol{x}^l \neq \mathbf{0}^l$, then in the graph $([k], E^{\mathrm{Pen}}_{\boldsymbol{X}} \cup E^{\mathrm{Com}}_{\boldsymbol{X}})$ we must have $1 \to^* l$, i.e., there must be a path in this graph from component 1 to all components l for which $\boldsymbol{x}^l$ is non-zero.

We denote by $\mathscr{X} \subset \mathscr{R}$ the set of all run-like recursive switching flows on G. We note for any G that we always have $\mathbf{0} \in \mathscr{X}$. We can show $\boldsymbol{X} \in \mathscr{F}$ is run-like if and only if $\exists t \in \mathbb{N}, \boldsymbol{X} = \boldsymbol{Run}(G,t)$.

We now introduce "unit vectors" for this space, we write $\boldsymbol{u}_e^l \in \mathscr{F}_l$ for the vector where $u_e^l = 1$ and for all other $e' \in E_l \cup F_l$ with $e' \neq e$ that $u_e^l = 0$. We then write $\boldsymbol{U}_{i,e} \in \mathscr{F}$ for the sequence of k vectors $(\mathbf{0}^1, \ldots, \mathbf{0}^{i-1}, \boldsymbol{u}_e^i, \mathbf{0}^{i+1}, \ldots, \mathbf{0}^k)$ where the i'th vector is $\boldsymbol{u}_e^i$ and for $i \neq j \in [k]$ that the j'th vector is the all-zero $\mathbf{0}^j$. We may naturally define the notion of addition on $\mathscr{F}$ and we define the notion of subtraction $\boldsymbol{X} - \boldsymbol{U}_{i,e}$ in the natural way whenever $x_e^i > 0$, i.e., the result of the subtraction remains in $\mathbb{N}$ for every coordinate, subtraction is undefined where this isn't the case. We write $\mathscr{U} := \{\boldsymbol{U}_{i,e} \mid i \in [k], e \in E_i \cup F_i\}$ for the set of all unit vectors.

Given a run-like recursive switching flow, $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{X}$, we say that $\boldsymbol{X}$ is *complete* if it is the case that $1 \in K_{\boldsymbol{X}}$, i.e., the current vertex $d_{\boldsymbol{x}^1}^1$ of $\boldsymbol{x}^1$ is an exit of G^1. We say $\boldsymbol{X}$ is *lassoed* when $E_{\boldsymbol{X}}^{\text{Pen}}$ forms a "lasso", meaning a directed line ending in one directed cycle, as described earlier. We note that being complete and lassoed are mutually exclusive, because either $1 \in K_{\boldsymbol{X}}$ or $1 \in J_{\boldsymbol{X}}$, but not both.

Lemma 4.1. *Let G be an instance of Recursive Arrival, and let $\boldsymbol{X} \in \mathscr{X}$ be a run-like recursive switching flow on G. Then if $\boldsymbol{X}$ is neither complete nor lassoed, then there exists exactly one $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $(\boldsymbol{X} + \boldsymbol{U}_{i,e})$ is a run-like recursive switching flow. Otherwise, if $\boldsymbol{X}$ is either complete or lassoed, then there exists no such $\boldsymbol{U}_{i,e}$.*

Proof (Sketch). We shall show that for any $\boldsymbol{X}$ which is neither complete nor lassoed, we are able to give unique i and e as a function of $\boldsymbol{X}$. Viewing $\boldsymbol{X}$ as a "hypothetical run" to some time we use $J_{\boldsymbol{X}}$ as our "call stack" at this time and use that to determine the edge to increment.

1. If $d_{\boldsymbol{X}}^{j_{(r_{\boldsymbol{X}}+1)}} \in \mathsf{Sor}_{j_{(r_{\boldsymbol{X}}+1)}}$, then the "current component" is at a switching node and we take the edge given by our switching order. We note that this includes the case where $d_{\boldsymbol{X}}^{j_{(r_{\boldsymbol{X}}+1)}} = o_{j_{(r_{\boldsymbol{X}}+1)}}$, i.e. there is a call pending to a new component.
2. If $j_{(r_{\boldsymbol{X}}+1)} \in K_{\boldsymbol{X}}$, then we can resolve the pending call in component $j_{r_{\boldsymbol{X}}}$ and increment the summary edge in $F_{j_{r_{\boldsymbol{X}}}}$ corresponding to exit $d_{\boldsymbol{X}}^{(j_{\boldsymbol{X}}+1)}$.

We can show that this is the unique choice in these cases through elimination, making use of the definitions of component, recursive, and run-like switching flows. □

We define the *completed call count* as the function $CC : \mathscr{F} \times [k] \to \mathbb{N}$ which counts how many times a given component has been crossed in a given flow, defined for $\boldsymbol{X} \in \mathscr{F}$ and $l \in [k]$ as follows:

$$CC(\boldsymbol{X}, l) := \sum_{i \in [k]} \sum_{\{b \in B_i \mid Y_i(b) = l\}} \sum_{f \in F_{b,i}} x_f^i$$

Lemma 4.2. *Let G be an instance of Recursive Arrival, and let $\boldsymbol{X} \in \mathscr{X}$ be a run-like recursive switching flow on G. If $\boldsymbol{X}$ is non-zero then there exists a unique $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $(\boldsymbol{X} - \boldsymbol{U}_{i,e}) \in \mathscr{X}$ is a run-like recursive switching flow. Otherwise, if $\boldsymbol{X}$ is all-zero, then no such $\boldsymbol{U}_{i,e}$ exists.*

Proof (Sketch). We shall show for non-zero $\boldsymbol{X}$ the following choice is the unique value for i, and then e can be determined using the last-used-edge graph in component i, as is the case for non-recursive switching flows. Viewing $\boldsymbol{X}$ as a "hypothetical run" to some time we use $J_{\boldsymbol{X}}$ as our "call stack" at this time and use that to determine the edge to decrement.

- If $\boldsymbol{x}^{j_{(r_{\boldsymbol{X}}+1)}} > \mathbf{0}^{j_{(r_{\boldsymbol{X}}+1)}}$ and $CC(\boldsymbol{X}, j_{(r_{\boldsymbol{X}}+1)}) = 0$ then we decrement inside the "current component" as the pending-call in component $j_{r_{\boldsymbol{X}}}$ is the only call made.

- Otherwise, we take $i = j_{r_{\boldsymbol{X}}}$. Where, since we have either $CC(\boldsymbol{X}, j_{(r_{\boldsymbol{X}}+1)}) \geq 1$ or $\boldsymbol{x}^{j_{(r_{\boldsymbol{X}}+1)}} = \boldsymbol{0}^{j_{(r_{\boldsymbol{X}}+1)}}$ the current call from $j_{r_{\boldsymbol{X}}}$ to $j_{(r_{\boldsymbol{X}}+1)}$ is either made elsewhere and thus we cannot alter the component flow in $j_{(r_{\boldsymbol{X}}+1)}$ without affecting the edge traversed on these other calls or the flow in $j_{(r_{\boldsymbol{X}}+1)}$ is zero, in which case we step back from the final pending-call to it.

This can be shown to be the unique choice in each case through elimination. □

We define the function $Val : \mathscr{F} \to \mathbb{N}$ as: $Val((\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k)) := \sum_{i \in [k]} \sum_{e \in E_i \cup F_i} x^i_e$. This function sums all values across all vectors of the tuple. We note that for any flow $\boldsymbol{X} \in \mathscr{F}$ and any $i \in [k]$ and $e \in E_i \cup F_i$ that we have $Val(\boldsymbol{X} + \boldsymbol{U}_{i,e}) = Val(\boldsymbol{X}) + 1$ and that when defined (i.e. $x^i_e > 0$) that $Val(\boldsymbol{X} - \boldsymbol{U}_{i,e}) = Val(\boldsymbol{X}) - 1$.

Recall Proposition 2.3 regarding non-recursive Arrival switching graphs, and in particular the fixed polynomial p which that proposition asserts the existence of. We say a recursive switching flow $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{X}$ is *finished* if it satisfies one of the following conditions:

1. $\boldsymbol{X}$ is complete, i.e, $1 \in K_{\boldsymbol{X}}$, or, the current vertex $d^1_{\boldsymbol{X}}$ of $\boldsymbol{x}^1$ is an exit in Ex_1.
2. $\boldsymbol{X}$ is lassoed, i.e., $1 \notin K_{\boldsymbol{X}}$ and $j_{(r_{\boldsymbol{X}}+1)} \in J_{\boldsymbol{X}}$, or, the edges of $E^{\text{Pen}}_{\boldsymbol{X}}$ form a lasso.
3. $\boldsymbol{X}$ is *just-overflowing*, which we define as follows: $1 \notin K_{\boldsymbol{X}}$, and there exists some unique $l \in [k]$, and unique $e = (u, d^l_{\boldsymbol{X}}) \in E_l \cup F_l$ with $x^l_e = 2^{\mathrm{p}(|V_l|)} + 1$, i.e., there is some unique component, l, and edge, e, incoming to its current vertex, $d^l_{\boldsymbol{X}}$, with a "just-excessively large" value in the flow $\boldsymbol{X}$.

We say the flow is *post-overflowing* if $1 \notin K_{\boldsymbol{X}}$, and there exists some $l \in [k]$, with $d^l_{\boldsymbol{X}}$ the current-vertex of $\boldsymbol{x}^l$, and some $e = (u, v) \in E_l \cup F_l$ satisfying at least one of: A) $x^l_e = 2^{\mathrm{p}(|V_l|)} + 1$ and $v \neq d^l_{\boldsymbol{X}}$; B) $x^l_e > 2^{\mathrm{p}(|V_l|)} + 1$. We note that by repeatedly applying Lemma 4.2 to a post-overflowing run-like recursive switching flow we must eventually find some finished just-overflowing run-like recursive switching flow.

We introduce the notation $\mathscr{F}^N \subseteq \mathscr{F}$ to be the restriction to tuples in which in every vector each coordinate is less than or equal to some $N \in \mathbb{N}$. Thus $\mathscr{F}^N$ is finite, and any element $\boldsymbol{X} \in \mathscr{F}^N$ can be represented using at most $(\sum_{i=1}^k |E_i \cup F_i|) \cdot \log_2(N)$ bits. For all our subsequent results taking $N := 2^{\mathrm{p}(\max_l |V_l|)} + 1$ will be sufficient, noting this means elements of $\mathscr{F}^N$ are represented using a polynomial number of bits in our input size.

Theorem 4.3. *The* `Recursive Arrival` *problem is in* $\mathsf{NP} \cap \mathsf{coNP}$ *and* $\mathsf{UP} \cap \mathsf{coUP}$.

Proof (Sketch). The proof follows from a series of lemmas given in the full version. These show:

- For any instance of `Recursive Arrival`, G, there is a (unique) $\boldsymbol{X} \in \mathscr{F}^N$ which is a finished run-like recursive switching flow;
- Given any $\boldsymbol{X} \in \mathscr{F}^N$ we can verify whether or not $\boldsymbol{X}$ is a finished run-like recursive switching flow in P-time;
- Given any $\boldsymbol{X} \in \mathscr{F}^N$ which is a finished run-like recursive switching flow, we can determine whether or not G terminates and if it does terminate at which exit in Ex_1 it does so.

□

4.1 Containment in UEOPL

Given the previous results, we may consider a search version of `Recursive Arrival` as follows:

`Search Recursive Arrival`	
Instance:	A Recursive Arrival graph $(G^1, \ldots, G^k)$
Problem:	Compute the unique finished run-like recursive switching flow $(\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{F}$ on G

In the appendix, we show that this problem is total and hence lies in TFNP. We show containment in the total search complexity class UEOPL defined by Fearnley et al. [12], as problems polynomial time many-one search reducible to `UniqueEOPL`, which is defined as follows:

`UniqueEOPL [12]`

Instance: Given boolean circuits $S, P : \{0,1\}^n \to \{0,1\}^n$ such that $P(0^n) = 0^n \neq S(0^n)$ and a boolean circuit $V : \{0,1\}^n \to \{0,1,\dots,2^m - 1\}$ such that $V(0^n) = 0$

Problem: Compute one of the following:

(U1) A point $x \in \{0,1\}^n$ such that $P(S(x)) \neq x$.

(UV1) A point $x \in \{0,1\}^n$ such that $x \neq S(x)$, $P(S(x)) = x$, and $V(S(x)) \leq V(x)$.

(UV2) A point $x \in \{0,1\}^n$ such that $S(P(x)) \neq x \neq 0^n$.

(UV3) Two points $x, y \in \{0,1\}^n$, such that $x \neq y$, $x \neq S(x)$, $y \neq S(y)$, and either $V(x) = V(y)$ or $V(x) < V(y) < V(S(x))$.

We may interpret an instance of `UniqueEOPL` as describing an exponentially large directed graph in which our vertices are points $x \in \{0,1\}^n$ and each vertex has both in-degree and out-degree bounded by at most one. Edges are described by the circuits S, P, for a fixed vertex $x \in \{0,1\}^n$ there is an outgoing edge from x to $S(x)$ if and only if $P(S(x)) = x$ and an incoming edge to x from $P(x)$ if and only if $S(P(x)) = x$. We are given that 0^n is a point with an outgoing edge but no incoming edge or the "start of the line". We also have an "odometer" function, V, which has a minimal value at 0^n. We assume our graph has the set-up of a single line $0^n, S(0^n), S(S(0^n)), \dots$ along which the function V strictly increases, with some "isolated points" where $x = S(x) = P(x)$. There are four types of solutions that can be returned, representing:

(U1) a point which is an "end of the line", with an incoming edge but no outgoing edge.

(UV1) a violation of the assumption that valuation V strictly increases along a line, since $V(x) \not< V(S(x))$.

(UV2) a violation of the assumption there is a single line, since x is the start of a line, but it is not 0^n, thus it starts a distinct line.

(UV3) a violation of one of the assumptions, however, in a more nuanced way. We can assume that $P(S(x)) = x$ and $P(S(y)) = y$, else they'd constitute a (UV1) example too, thus neither x nor y is isolated and both have an outgoing edge. If x and y were on the same line, then either $S(\dots S(S(x))) = y$ or $S(\dots S(y)) = x$ by doing this iteration we'd eventually find some $z \in \{0,1\}^n$ where $V(z) \not< V(S(z))$, violating (UV1). However, if x and y are on different lines, then that would imply the existence of two distinct lines, violating (UV2). Thus, a (UV3) violation is a short proof of existence of a (UV1) or (UV2) violation elsewhere in the instance.

For our reduction, our space will be made up of all possible flows $(\boldsymbol{x}^1, \dots, \boldsymbol{x}^k) \in \mathscr{F}^N$ and our line will be made up of those arising from distinct $\boldsymbol{Run}(G,t)$'s, each step increasing in t until we reach a finished flow, with all other vectors being isolated. A type (U1) solution will correspond to a finished run-like recursive switching flow, and we will show our instance has no (UV1-3) solutions, thus our computed solution to `UniqueEOPL` will be a solution to `Search Recursive Arrival`.

Given any flow $\boldsymbol{X} \in \mathscr{F}$ we can verify whether or not $\boldsymbol{X}$ is a run-like recursive switching flow (i.e. $\boldsymbol{X} \in \mathscr{X} \subset \mathscr{F}$). We will use this fact in our definitions of functions $Adv : \mathscr{F} \to \mathscr{F}$ and $Prev : \mathscr{F} \to \mathscr{F}$.

Our function Adv on some value $\boldsymbol{X} := (\boldsymbol{x}^1, \dots, \boldsymbol{x}^k) \in \mathscr{F}$ is defined by the following sequence:

1. If $\boldsymbol{X} \notin \mathscr{X}$ then we take $Adv(\boldsymbol{X}) = \boldsymbol{X}$.

2. Else if $\boldsymbol{X} \in \mathscr{X}$ is either finished or post-overflowing then we take $Adv(\boldsymbol{X}) = \boldsymbol{X}$.
3. Otherwise, take $Adv(\boldsymbol{X}) = \boldsymbol{X} + \boldsymbol{U}_{i,e}$, for the unique $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $\boldsymbol{X} + \boldsymbol{U}_{i,e} \in \mathscr{X}$ (Lemma 4.1).

We note by this process that if $Adv(\boldsymbol{X}) \neq \boldsymbol{X}$, then $Val(Adv(\boldsymbol{X})) = Val(\boldsymbol{X}) + 1$, since we have incremented exactly one edge in exactly one vector. Hence, this is consistent with our odometer. We may also define the operation $Prev : \mathscr{F} \to \mathscr{F}$ analogously on some value $\boldsymbol{X} := (\boldsymbol{x}^1, \ldots, \boldsymbol{x}^k) \in \mathscr{F}$. Taking $Prev(\boldsymbol{X}) = \boldsymbol{X}$ whenever: $\boldsymbol{X} \notin \mathscr{X}$; $\boldsymbol{X} = \mathbf{0}$, or; $\boldsymbol{X}$ is post-overflowing. Otherwise, taking $Prev(\boldsymbol{X}) = \boldsymbol{X} - \boldsymbol{U}_{i,e}$, for the unique $\boldsymbol{U}_{i,e} \in \mathscr{U}$ such that $\boldsymbol{X} - \boldsymbol{U}_{i,e} \in \mathscr{X}$ (Lemma 4.2). Observe that, for any non-zero $\boldsymbol{X} \in \mathscr{X}$, that $Adv(Prev(\boldsymbol{X})) = \boldsymbol{X}$, and, for any $\boldsymbol{X} \in \mathscr{X}$, if we have $Prev(Adv(\boldsymbol{X})) \neq \boldsymbol{X}$, then $\boldsymbol{X}$ must be finished.

Theorem 4.4. *The* `Search-Recursive Arrival` *is in* UEOPL.

Proof (Sketch). We will give a polynomial-time search reduction from `Search Recursive Arrival` to the `UniqueEOPL` problem. We compute boolean circuits S, P and V which will be given by the restriction of the functions *Adv*, *Prev*, and *Val* to the domain $\mathscr{F}^N$. This process involves computing membership of $\mathscr{X}$ and then computing the unique values i and e given by Lemmas 4.1 and 4.2 for *Adv* and *Prev* respectively. We can then show that the only UEOPL solution is of type (U1) and is a run-like recursive switching flow, which is a solution we are looking for. □

5 Conclusions

We have shown that `Recursive Arrival` is contained in many of the same classes as the standard `Arrival` problem. While we have shown P-hardness for `Recursive Arrival`, whether or not `Arrival` is P-hard remains open.

Let us note that the way we have chosen to generalise Arrival to the recursive setting uses one of two possible natural choices for its semantics. Namely, it assumes a “local” semantics, meaning that the current switch position for each component on the call stack is maintained as part of the current state. An alternative “global” semantics would instead consider the switch position of each component as a “global variable”. In such a model all switch positions would start in an initial position, and as the run progresses the switch positions would persist between, and be updated during, different calls to the same component. It is possible to show (a result we have not included in this paper) that such a “global" formulation immediately results in PSPACE-hardness of reachability and termination problems.

As mentioned in the introduction, a stochastic version of `Arrival`, in which some nodes are switching nodes whereas other nodes are chance (probabilistic) nodes with probabilities on outgoing transitions, has already been studied in [19], building on the work of [12] which generalises `Arrival` by allowing switching and player-controlled nodes. There is extensive prior work on RMCs and RMDPs, with many known decidability/complexity results (see, e.g., [9, 10]). It would be natural to ask similar computational questions for the generalisation of RMCs and RMDPs to a recursive Arrival model combining switching nodes with chance (probabilistic) nodes and controlled/player nodes.

Finally, we note that Fearnley et al. also defined a P-hard generalisation of `Arrival` in [12] which uses “succinct switching orders” to succinctly encode an exponentially larger switch graph. We will refer to this problem as `Succinct Arrival`. We don’t know whether there are any P-time reduction, in either direction, between `Recursive Arrival` and `Succinct Arrival`. It has been observed[1] that the results of [14] imply that both `Arrival` and `Succinct Arrival` are P-time reducible to the `Tarski` problem

[1]Personal communication from Kousha Etessami and Mihalis Yannakakis.

defined in [7]. `Succinct Arrival` is also contained in UEOPL by the same arguments as for `Arrival`. We do not currently know whether `Recursive Arrival` is P-time reducible to `Tarski`.

Acknowledgements. Thanks to my PhD supervisor Kousha Etessami for his support.

References

[1] Rajeev Alur, Kousha Etessami & Mihalis Yannakakis (2001): *Analysis of Recursive State Machines*. In Gérard Berry, Hubert Comon & Alain Finkel, editors: *Computer Aided Verification*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 207–220, doi:10.5555/647770.734260.

[2] David Auger, Pierre Coucheney & Loric Duhaze (2022): *Polynomial Time Algorithm for ARRIVAL on Tree-like Multigraphs*. In: *International Symposium on Mathematical Foundations of Computer Science*, doi:10.48550/arXiv.2204.13151.

[3] Ahmed Bouajjani, Javier Esparza & Oded Maler (1997): *Reachability analysis of pushdown automata: Application to model-checking*. In Antoni Mazurkiewicz & Józef Winkowski, editors: *CONCUR '97: Concurrency Theory*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 135–150, doi:10.1007/3-540-63141-0_10.

[4] Anne Condon (1992): *The Complexity of Stochastic Games*. *Inf. Comput.* 96(2), pp. 203–224, doi:10.1016/0890-5401(92)90048-K.

[5] Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jíri Matoušek & Emo Welzl (2017): *A Journey through Discrete Mathematics: A Tribute to Jiri Matousek*, chapter Arrival: A zero-player graph game in $\mathsf{NP} \cap \mathsf{coNP}$, pp. 367–374. Springer, doi:10.1007/978-3-319-44479-6_14.

[6] Javier Esparza, Antonin Kucera & Richard Mayr (2006): *Model Checking Probabilistic Pushdown Automata*. *Logical Methods in Computer Science* Volume 2, Issue 1, doi:10.2168/LMCS-2(1:2)2006.

[7] Kousha Etessami, Christos H. Papadimitriou, Aviad Rubinstein & Mihalis Yannakakis (2020): *Tarski's Theorem, Supermodular Games, and the Complexity of Equilibria*. In Thomas Vidick, editor: *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, *LIPIcs* 151, pp. 18:1–18:19, doi:10.4230/LIPIcs.ITCS.2020.18.

[8] Kousha Etessami & Mihalis Yannakakis (2008): *Recursive Concurrent Stochastic Games*. *CoRR* abs/0810.3581, doi:10.48550/arXiv.0810.3581. arXiv:0810.3581.

[9] Kousha Etessami & Mihalis Yannakakis (2009): *Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations*. *J. ACM* 56(1), pp. 1:1–1:66, doi:10.1145/1462153.1462154.

[10] Kousha Etessami & Mihalis Yannakakis (2015): *Recursive Markov Decision Processes and Recursive Stochastic Games*. *J. ACM* 62(2), pp. 11:1–11:69, doi:10.1145/2699431.

[11] John Fearnley, Martin Gairing, Matthias Mnich & Rahul Savani (2021): *Reachability Switching Games*. *Log. Methods Comput. Sci.* 17(2), doi:10.23638/LMCS-17(2:10)2021.

[12] John Fearnley, Spencer Gordon, Ruta Mehta & Rahul Savani (2019): *Unique End of Potential Line*. In: *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, *LIPIcs* 132, pp. 56:1–56:15, doi:10.4230/LIPIcs.ICALP.2019.56.

[13] Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubácek, Karel Král, Hagar Mosaad & Veronika Slívová (2018): *ARRIVAL: Next Stop in CLS*. In: *45th International Colloquium on Automata, Languages, and Programming*, *LIPIcs* 107, pp. 60:1–60:13, doi:10.4230/LIPIcs.ICALP.2018.60.

[14] Bernd Gärtner, Sebastian Haslebacher & Hung P. Hoang (2021): *A Subexponential Algorithm for ARRIVAL*. In: *48th International Colloquium on Automata, Languages, and Programming*, *LIPIcs* 198, pp. 69:1–69:14, doi:10.4230/LIPIcs.ICALP.2021.69.

[15] Raymond Greenlaw, H. James Hoover & Walter L. Ruzzo (1995): *Limits to Parallel Computation*. Oxford University Press, doi:10.1093/oso/9780195085914.001.0001.

[16] Marcin Jurdzinski (1998): *Deciding the Winner in Parity Games is in* $\mathsf{UP} \cap \mathsf{coUP}$. *Inf. Process. Lett.* 68(3), pp. 119–124, doi:10.1016/S0020-0190(98)00150-1.

[17] C. S. Karthik (2017): *Did the train reach its destination: The complexity of finding a witness*. *Information Processing Letters* 121, pp. 17–21, doi:10.1016/j.ip1.2017.01.004.

[18] Graham Manuell (2021): *A simple lower bound for ARRIVAL*. *CoRR* abs/2108.06273, doi:10.48550/arXiv.2108.06273.

[19] Thomas Webster (2022): *The Stochastic Arrival Problem*. In Anthony W. Lin, Georg Zetzsche & Igor Potapov, editors: *Reachability Problems*, Springer, pp. 93–107, doi:10.1007/978-3-031-19135-0_7.

[20] Uri Zwick & Mike Paterson (1996): *The Complexity of Mean Payoff Games on Graphs*. *Theor. Comput. Sci.* 158(1&2), pp. 343–359, doi:10.1016/0304-3975(95)00188-3.

On the Descriptive Complexity of Groups without Abelian Normal Subgroups

(Extended Abstract)

Joshua A. Grochow
Department of Computer Science University of Colorado Boulder, CO USA
Department of Mathematics University of Colorado Boulder, CO USA
jgrochow@colorado.edu

Michael Levet
Department of Computer Science, College of Charleston, SC USA
levetm@cofc.edu

In this paper, we explore the descriptive complexity theory of finite groups by examining the power of the second Ehrenfeucht–Fraïssé bijective pebble game in Hella's (*Ann. Pure Appl. Log.*, 1989) hierarchy. This is a Spoiler–Duplicator game in which Spoiler can place up to two pebbles each round. While it trivially solves graph isomorphism, it may be nontrivial for finite groups, and other ternary relational structures. We first provide a novel generalization of Weisfeiler–Leman (WL) coloring, which we call *2-ary* WL. We then show that the 2-ary WL is equivalent to the second Ehrenfeucht–Fraïssé bijective pebble game in Hella's hierarchy.

Our main result is that, in the pebble game characterization, only $O(1)$ pebbles and $O(1)$ rounds are sufficient to identify all groups without Abelian normal subgroups (a class of groups for which isomorphism testing is known to be in P; Babai, Codenotti, & Qiao, ICALP 2012). In particular, we show that within the first few rounds, Spoiler can force Duplicator to select an isomorphism between two such groups at each subsequent round. By Hella's results (*ibid.*), this is equivalent to saying that these groups are identified by formulas in first-order logic with generalized 2-ary quantifiers, using only $O(1)$ variables and $O(1)$ quantifier depth.

1 Introduction

Descriptive complexity theory studies the relationship between the complexity of describing a given problem in some logic, and the complexity of solving the problem by an algorithm. When the problems involved are isomorphism problems, Immerman and Lander [44] showed that complexity of a logical sentence describing the isomorphism type of a graph was essentially the same as the Weisfeiler–Leman coloring dimension of that graph, and the complexity of an Ehrenfeucht–Fraïssé pebble game (see also [17]).

It is a well-known open question whether there is a logic that exactly captures the complexity class P on unordered (unlabeled) structures; on ordered structures such a logic was given by Immerman [43] and Vardi [67]. The difference between these two settings is essentially the GRAPH CANONIZATION problem, whose solution allows one to turn an unordered graph into an ordered graph in an isomorphism-preserving way.

One natural approach in trying to capture P on unordered structures is thus to attempt to extend first-order logic FO by generalized quantifiers (c.f., Mostowski [60] and Lindstrom [57]) in the hopes that the augmented logics can characterize finite graphs up to isomorphism, thus reducing the unordered case to the previously solved ordered case. A now-classical approach, initiated by Immerman [43], was to augment fixed-point logic with counting quantifiers, which can be analyzed in terms of an equivalence induced by (variable confined) fragments of first-order logic with counting. However, Cai, Fürer, & Immerman [17] showed that $\mathsf{FO}+\mathsf{LFP}$ plus counting does not capture P on finite graphs. More generally, Flum & Grohe have characterized when FO plus counting captures P on unordered structures [26].

The approach of Cai, Fürer, & Immerman (ibid., see also [44]) was to prove a three-way equivalence: between (1) counting logics, (2) the higher-dimensional Weisfeiler–Leman coloring procedure, and (3) Ehrenfeucht–Fraïssé pebble games. Ehrenfeucht–Fraïssé pebble games [24, 27] have long been an important tool in proving the inexpressibility of certain properties in various logics; in this case, they used such games to show that the logics could not express the difference between certain pairs of non-isomorphic graphs. Consequently, Cai, Fürer, & Immerman ruled out the Weisfeiler–Leman algorithm as a polynomial-time isomorphism test for graphs, which resolved a long-standing open question in isomorphism testing. Nonetheless, the Weisfeiler–Leman coloring procedure is a key subroutine in many algorithms for Graph Isomorphism, including Babai's quasi-polynomial-time algorithm [4]. It is thus interesting to study its properties and its distinguishing power.

While the result of Cai, Fürer, & Immerman ruled out Weisfeiler–Leman as a polynomial-time isomorphism test for graphs, for *groups* it remains an interesting open question. The general WL procedure for groups was introduced by Brachter & Schweitzer [12] and has been studied in several papers since then [13, 30]. Outside the scope of WL, it is known that Group Isomorphism is AC^0-reducible to Graph Isomorphism, and there is no AC^0 reduction in the opposite direction [19]. For this and other reasons, group isomorphism is believed to be the easier of the two problems, so it is possible that WL—and more generally, tools from descriptive complexity—could yield stronger results for groups than for graphs.

On graphs, which are binary relational structures, if Spoiler is allowed to pebble two elements per turn, then Spoiler can win on any pair of non-isomorphic graphs. However, groups are ternary relational structures (the relation is $\{(a,b,c) : ab = c\}$), so such a game may yield nontrivial insights into the descriptive complexity of finite groups. Hella [41, 42] introduced such games in a more general context, and showed that allowing Spoiler to pebble q elements per round corresponded to the generalized q-ary quantifiers of Mostowski [60] and Lindstrom [57]. When $q = 1$, Hella shows that this pebble game is equivalent in power to the FO plus counting logics mentioned above. Our focus in this paper is to study the power of the $q = 2$-ary game for identifying finite groups.

Main Results. In this paper, we initiate the study of Hella's 2-ary Ehrenfeucht–Fraïssé-style pebble game, in the setting of groups. Our main result is that this pebble game efficiently characterizes isomorphism in a class of groups for which isomorphism testing is known to be in P, but only by quite a nontrivial algorithm (see remark below). The full version of this paper appears on arXiv [29].

Theorem 1.1. *Let G be a group with no Abelian normal subgroups (a.k.a. Fitting-free or semisimple), and let H be arbitrary. If $G \not\cong H$, then Spoiler has a winning strategy in the Ehrenfeucht–Fraïssé game at the second level of Hella's hierarchy, using 9 pebbles and $O(1)$ rounds.*

In proving Thm. 1.1, we show that with the use of only a few pebbles, Spoiler can effectively force Duplicator to select an isomorphism of G and H. We contrast this with the setting of Weisfeiler–Leman (which is equivalent to the 1-ary pebble game), for which the best upper bound we have on the WL-dimension is the trivial bound of $\log n$. Furthermore, we do not have any lower bounds on the WL-dimension for semisimple groups.

Remark 1.2. Every group G can be written as an extension of its solvable radical $\mathrm{Rad}(G)$ by the quotient

$G/\mathrm{Rad}(G)$, which does not have Abelian normal subgroups. As such, the latter class of groups is quite natural, both group-theoretically and computationally. Computationally, it has been used in algorithms for general finite groups both in theory (e.g., [5, 6]) and in practice (e.g., [18]). Isomorphism testing in this family of groups can be solved efficiently in practice [18], and is known to be in P through a series of two papers [7, 8].

In Section 3 we also complete the picture by giving a Weisfeiler–Leman-style coloring procedure and showing that it corresponds precisely to Hella's q-ary pebble games and q-ary generalized Lindstrom quantifiers [57]. When the groups are given by their multiplication tables, this procedure runs in time $n^{\Theta(\log^2 n)}$ by reduction to GRAPH ISOMORPHISM. We note that Hella's results deal with infinitary logics [41, 42]. However, as we are dealing with finite groups, the infinitary quantifiers and connectives are not necessary (see the discussion in [42], right above Theorem 5.3).

Further Related Work. Despite the fact that Weisfeiler–Leman is insufficient to place GRAPH ISOMORPHISM (GI) into PTIME, it remains an active area of research. For instance, Weisfeiler–Leman is a key subroutine in Babai's quasipolynomial-time GI algorithm [4]. Furthermore, Weisfeiler–Leman has led to advances in simultaneously developing both efficient isomorphism tests and the descriptive complexity theory for finite graphs- see for instance, [32, 38, 49, 50, 34, 36, 35, 51, 1, 2, 64]. Weisfeiler–Leman also has close connections to the Sherali–Adams hierarchy in linear programming [37].

The complexity of the GROUP ISOMORPHISM (GPI) problem is a well-known open question. In the Cayley (multiplication) table model, GPI belongs to $\mathsf{NP} \cap \mathsf{coAM}$. The generator-enumerator algorithm, attributed to Tarjan in 1978 [59], has time complexity $n^{\log_p(n)+O(1)}$, where n is the order of the group and p is the smallest prime dividing n. This bound has escaped largely unscathed: Rosenbaum [63] (see [55, Sec. 2.2]) improved this to $n^{(1/4)\log_p(n)+O(1)}$. And even the impressive body of work on practical algorithms for this problem, led by Eick, Holt, Leedham-Green and O'Brien (e. g., [11, 25, 10, 18]) still results in an $n^{\Theta(\log n)}$-time algorithm in the general case (see [70, Page 2]). In the past several years, there have been significant advances on algorithms with worst-case guarantees on the serial runtime for special cases of this problem including Abelian groups [47, 68, 65], direct product decompositions [69, 48], groups with no Abelian normal subgroups [7, 8], coprime and tame group extensions [54, 62, 9, 31], low-genus p-groups and their quotients [56, 15], Hamiltonian groups [20], and groups of almost all orders [23].

Key motivation for GPI is due to its close relation to GI. In the Cayley (verbose) model, GPI reduces to GI [71], while GI reduces to the succinct GPI problem [40, 58] (recently simplified [39]). In light of Babai's breakthrough result that GI is quasipolynomial-time solvable [4], GPI in the Cayley model is a key barrier to improving the complexity of GI. Both verbose GPI and GI are considered to be candidate NP-intermediate problems, that is, problems that belong to NP, but are neither in P nor NP-complete [53]. There is considerable evidence suggesting that GI is not NP-complete [66, 16, 45, 4, 52, 3]. As verbose GPI reduces to GI, this evidence also suggests that GPI is not NP-complete. It is also known that GI is strictly harder than GPI under AC^0 reductions [19].

While the descriptive complexity of graphs has been extensively studied, the work on the descriptive complexity of groups is scant compared to the algorithmic literature on GROUP ISOMORPHISM (GPI). There has been work relating first order logics and groups [61], as well as work examining the descriptive complexity of finite abelian groups [28]. Recently, Brachter & Schweitzer [12] introduced three variants of Weisfeiler–Leman for groups, including corresponding logics and pebble games. These pebble games correspond to the first level of Hella's hierarchy [41, 42]. In particular, Brachter & Schweitzer showed that 3-dimensional Weisfeiler–Leman can distinguish p-groups arising from the CFI graphs [17] via Mekler's construction [58], suggesting that $\mathsf{FO} + \mathsf{LFP} + \mathsf{C}$ may indeed capture PTIME on groups. Determining whether even $o(\log n)$-dimensional Weisfeiler–Leman can resolve GPI is an open question.

The use on Weisfeiler–Leman for groups is quite new. To the best of our knowledge, using Weisfeiler–Leman for GROUP ISOMORPHISM testing was first attempted by Brooksbank, Grochow, Li, Qiao, & Wilson [14]. Brachter & Schweitzer [12] subsequently introduced three variants of Weisfeiler–Leman for groups that more closely resemble that of graphs. In particular, Brachter & Schweitzer [12] characterized their algorithms in terms of logics and Ehrenfeucht–Fraïssé pebble games. The relationship between the works of Brachter & Schweitzer and Brooksbank, Grochow, Li, Qiao, & Wilson [14] is an interesting question.

In subsequent work, Brachter & Schweitzer [13] further developed the descriptive complexity of finite groups. They showed in particular that low-dimensional Weisfeiler–Leman can detect key group-theoretic invariants such as composition series, radicals, and quotient structure. Furthermore, they also showed that Weisfeiler–Leman can identify direct products in polynomial-time, provided it can also identify the indecomposable direct factors in polynomial-time. Grochow & Levet [30] extended this result to show that Weisfeiler–Leman can compute direct products in parallel, provided it can identify each of the indecomposable direct factors in parallel. Additionally, Grochow & Levet showed that constant-dimensional Weisfeiler–Leman can in a constant number of rounds identify coprime extensions $H \ltimes N$, where the normal Hall subgroup N is Abelian and the complement H is $O(1)$-generated. This placed isomorphism testing into L; the previous bound for isomorphism testing in this family was P [62]. Grochow & Levet also ruled out $\mathsf{FO} + \mathsf{LFP}$ as a candidate logic for capturing PTIME on finite groups, by showing that the count-free Weisfeiler–Leman algorithm cannot even identify Abelian groups in polynomial-time.

2 Preliminaries

We recall the bijective pebble game of Hella [41, 42], in the context of WL on graphs as that is likely more familiar to more readers. This game is often used to show that two graphs X and Y cannot be distinguished by k-WL. The game is an Ehrenfeucht–Fraïssé game, with two players: Spoiler and Duplicator. Each graph begins with $k+1$ pebbles, $p_1, \ldots, p_{k+1}$ for X and $p'_1, \ldots, p'_{k+1}$ for Y, which are placed beside the graphs. Each round proceeds as follows.

1. Spoiler chooses $i \in [k+1]$, and picks up pebbles p_i, p'_i.
2. We check the winning condition, which will be formalized later.[1]
3. Duplicator chooses a bijection $f : V(X) \to V(Y)$.
4. Spoiler places p_i on some vertex $v \in V(X)$. Then p'_i is placed on $f(v)$.

In a given round, let $v_1, \ldots, v_m$ be the vertices of X pebbled at the end of step 1 (in the list above), and let $v'_1, \ldots, v'_m$ be the corresponding pebbled vertices of Y. Spoiler wins precisely if the map $v_\ell \mapsto v'_\ell$ is not an isomorphism of the induced subgraphs $X[\{v_1, \ldots, v_m\}]$ and $Y[\{v'_1, \ldots, v'_m\}]$. Otherwise, at that point, Duplicator wins the game. Spoiler wins, by definition, at round 0 if X and Y do not have the same number of vertices. We note that X and Y are not distinguished by the first r rounds of k-WL if and only if Duplicator wins the first r rounds of the $(k+1)$-pebble game [41, 42, 17].

[1] In the literature, some authors check the winning condition at this point, and others check the winning condition at the end of each round. The choice merely has the effect of changing the number of required pebbles by at most 1 in ordinary WL, or at most q in the q-ary version, and changing the number of rounds by at most 1. We have chosen this convention for consistency with other works on WL specific to groups [12, 13, 30].

Hella [41, 42] exhibited a hierarchy of pebble games where, for $q \geq 1$, Spoiler could pebble a sequence of $1 \leq j \leq q$ elements $(v_1, \dots, v_j) \mapsto (f(v_1), \dots, f(v_j))$ in a single round; more formally, following the description above, in step 1, Spoiler picks up q pebbles $p_{i_1}, \dots, p_{i_q}$ and their partners $p'_{i_1}, \dots, p'_{i_q}$, with step 4 changed accordingly. The case of $q = 1$ corresponds to the case of Weisfeiler–Leman. As remarked by Hella [42, p. 6, just before §4], the q-ary game immediately identifies all relational structures of arity $\leq q$. For example, the $q = 2$ game on graphs solves GI: for if two graphs X and Y are non-isomorphic, then any bijection $f : V(X) \to V(Y)$ that Duplicator selects must map an adjacent pair of vertices u, v in X to a non-adjacent pair $f(u), f(v)$ in Y or vice-versa. Spoiler immediately wins by pebbling $(u, v) \mapsto (f(u), f(v))$. However, as groups are ternary relational structures (the relation being $\{(a,b,c) : a,b,c \in G, ab = c\}$), the $q = 2$ case can, at least in principle, be non-trivial on groups.

Brachter & Schweitzer [12] adapted Hella's [41, 42] pebble games in the $q = 1$ case to the setting of groups, obtaining three different versions. Their Version III involves reducing to graphs and playing the pebble game on graphs, so we don't consider it further here. Versions I and II are both played on the groups G and H directly.

Both versions are played identically as for graphs, with the only difference being the winning condition. We recall the following standard definitions in order to describe these winning conditions.

Definition 2.1. Let G, H be two groups. Given k-tuples $\overline{g} = (g_1, \dots, g_k) \in G^k$ and $\overline{h} = (h_1, \dots, h_k) \in H^k$, we say $(\overline{g}, \overline{h})$ ...

1. *...gives a well-defined map* if $g_i = g_j \Leftrightarrow h_i = h_j$ for all $i \neq j$;
2. ...are *partially isomorphic* or *give a partial isomorphism* if they give a well-defined map, and for all i, j, k we have $g_i g_j = g_k \Leftrightarrow h_i h_j = h_k$;
3. ...are *marked isomorphic* or *give a marked isomorphism* if it gives a well-defined map, and the map extends to an isomorphism $\langle g_1, \dots, g_k \rangle \to \langle h_1, \dots, h_k \rangle$.

Let $v_1, \dots, v_m$ be the group elements of G pebbled at the end of step 1, and let $v'_1, \dots, v'_m$ be the corresponding pebbled vertices of H. In Version I, Spoiler wins precisely if $(\overline{v}, \overline{v}')$ does not give a partial isomorphism, and in Version II Spoiler wins precisely if $(\overline{v}, \overline{v}')$ does not give a marked isomorphism.

Both Versions I and II may be generalized to allow Spoiler to pebble up to q group elements at a single round, for some $q \geq 1$. Mimicking the proof above for $q = 2$ for graphs, we have that $q = 3$ is sufficient to solve GPI in a single round. The distinguishing power, however, of the $q = 2$ game for groups remains unclear, and is the main subject of this paper. As we are interested in the round complexity, we introduce the following notation.

Definition 2.2 (Notation for pebbles, rounds, arity, and WL version)**.** Let $k \geq 2, r \geq 1$, $q \geq 1$, and $J \in \{I, II\}$. Denote (k,r)-WL$_J^q$ to be the k-pebble, r-round, q-ary Version J pebble game.

We refer to q as the *arity* of the pebble game, as it corresponds to the arity of generalized quantifiers[2] in a logic whose distinguishing power is equivalent to that of the game:

Remark 2.3 (Equivalence with logics with generalized 2-ary quantifiers)**.** Hella [41] describes the game (essentially the same as our description, but with no restriction on number of pebbles, and a transfinite number of rounds) for general q at the bottom of p. 245, for arbitrary relational structures. We restrict to the case of $q = 2$, a finite number of pebbles and rounds, and the (relational) language of groups. Hella proves that this game is equivalent to first-order logic with arbitrary q-ary equantifiers in [41, Thm. 2.5].

[2]As our focus in this paper is not on the viewpoint of generalized quantifiers, we refer the reader to [41] for details.

Observation 2.4. *In the* 2*-ary pebble game, we may assume that Duplicator selects bijections that preserve inverses.*

Proof. Suppose not. First, Duplicator must select bijections that preserve the identity, for if not, Spoiler pebbles $1_G \mapsto f(1) \neq 1_H$ and wins immediately. Next, let $f : G \to H$ be a bijection such that $f(g^{-1}) \neq f(g)^{-1}$. Spoiler pebbles $(g, g^{-1}) \mapsto (f(g), f(g^{-1}))$. Now $gg^{-1} = 1$, while $f(g)f(g^{-1}) \neq 1$. So Spoiler wins. □

We frequently use this observation without mention.

3 Higher-arity Weisfeiler-Leman-style coloring corresponding to higher arity pebble games

Given a k-tuple $\overline{x} = (x_1, \ldots, x_k) \in G^k$, a pair of distinct indices $i, j \in [k]$, and a pair of group elements y, z, we define $\overline{x}_{(i,j) \leftarrow (y,z)}$ to be the k-tuple $\overline{x}'$ that agrees with $\overline{x}$ on all indices besides i, j, and with $x'_i = y, x'_j = z$. If $i = j$, we require $y = z$, and we denote this $\overline{x}_{i \leftarrow y}$.

Finally, two graphs Γ_1, Γ_2, with edge-colorings $c_i \colon E(\Gamma_i) \to C$ to some color set C (for $i = 1, 2$) are color isomorphic if there is a graph isomorphism $\varphi \colon V(\Gamma_1) \to V(\Gamma_2)$ that also preserves colors, in the sense that $c_1((u, v)) = c_2((\varphi(u), \varphi(v))$ for all edges $(u, v) \in E(\Gamma_1)$.

Definition 3.1 (2-ary k-dimensional Weisfeiler-Leman coloring)**.** Let G, H be two groups of the same order, let $k \geq 1$. For all k-tuples $\overline{x}, \overline{y} \in G^k \cup H^k$:

- (Initial coloring, Version I) $\chi_0^{2,I}(\overline{x}) = \chi_0^{2,I}(\overline{y})$ iff $\overline{x}, \overline{y}$ are partially isomorphic.
- (Initial coloring, Version II) $\chi_0^{2,II}(\overline{x}) = \chi_0^{2,II}(\overline{y})$ iff $\overline{x}, \overline{y}$ have the same marked isomorphism type.
- (Color refinement) Given a coloring $\chi \colon G^k \cup H^k \to C$, the color refinement operator R defines a new coloring $R(\chi)$ as follows. For each k-tuple $\overline{x} \in G^k$ (resp., H^k), we define an edge-colored graph $\Gamma_{\overline{x},\chi,i,j}$. If $i = j$, it is the graph on vertex set $V(\Gamma_{\overline{x},\chi,i,i}) = G$ (resp., H) with all self-loops and no other edges, where the color of each self-loop (g, g) is $\chi(\overline{x}_{i \leftarrow g})$. If $i \neq j$, it is the complete directed graph with self-loops on vertex set G (resp., H), where the color of each edge (y, z) is $\chi(\overline{x}_{(i,j) \leftarrow (y,z)})$. For an edge-colored graph Γ, we use $[\Gamma]$ to denote its edge-colored isomorphism class. We then define
$$R(\chi)(\overline{x}) = \left(\chi(\overline{x}); [\Gamma_{\overline{x},\chi,1,1}], [\Gamma_{\overline{x},\chi,1,2}], \ldots, [\Gamma_{\overline{x},\chi,k-1,k}], [\Gamma_{\overline{x},\chi,k,k}]\right).$$
That is, the new color consists of the old color, as well as the tuple of $\binom{k+1}{2}$ edge-colored isomorphism types of the graphs $\Gamma_{\overline{x},\chi,i,j}$.

The refinement operator may be iterated: $R^t(\chi) := R(R^{t-1}(\chi))$, and we define the *stable refinement* of χ as $R^t(\chi)$ where the partition induced by $R^t(\chi)$ on $G^k \cup H^k$ is the same as that induced by $R^{t+1}(\chi)$. We denote the stable refinement by $R^\infty(\chi)$.

Finally, for $J \in \{I, II\}$ and all $r \geq 0$, we define $\chi_{r+1}^{2,J} = R(\chi_r^{2,J})$, and $\chi_\infty^{2,J} := R^\infty(\chi_0^{2,J})$.

Remark 3.2. Brachter & Schweitzer [12] introduced Versions I and II of 1-ary WL, which are equivalent up to a small additive constant in the WL-dimension [12] and $O(\log n)$ rounds [30]. For the purpose of comparison, we introduce Versions I and II of 2-ary WL. We will see later that only one additional round suffices in the 2-ary case (see Thm. 3.7). The differences in Versions I and II of WL (both the 1-ary and 2-ary variants) arise from whether the group is viewed as a structure with a ternary relational structure (Version I) or as a structure with a binary function (Version II).

Remark 3.3. Since it was one of our stumbling blocks in coming up with this generalized coloring, we clarify here how this indeed generalizes the usual 1-ary WL coloring procedure. In the 1-ary "oblivious" k-WL procedure (see [33, §5], equivalent to ordinary WL), the color of a k-tuple $\overline{x}$ is refined using its old color, together with a k-tuple of multisets

$$(\{\{\chi(x_{1\leftarrow y}) : y \in G\}\}, \{\{\chi(x_{2\leftarrow y}) : y \in G\}\}, \ldots, \{\{\chi(x_{k\leftarrow y}) : y \in G\}\}).$$

For each i, note that two multisets $\{\{\chi(x_{i\leftarrow y}) : y \in G\}\}$ and $\{\{\chi(x'_{i\leftarrow y}) : y \in G\}\}$ are equal iff the graphs $\Gamma_{\overline{x},\chi,i,i}$ and $\Gamma_{\overline{x}',\chi,i,i}$ are color-isomorphic. That is, edge-colored graphs with only self-loops and no other edges are essentially the same, up to isomorphism, as multisets. Our procedure generalizes this by also considering graphs with other edges, which (as we'll see in the proof of equivalence, which will appear in the full version) are used to encode the choice of 2 simultaneous pebbles by Spoiler in each move of the game.

Theorem 3.4. *Let G,H be two groups of order n, with $\overline{x} \in G^k, \overline{y} \in H^k$. Starting from the initial pebbling $x_i \mapsto y_i$ for all $i = 1, \ldots, k$, Spoiler has a winning strategy in the k-pebble, r-round, 2-ary Version J pebble game (for $J \in \{I, II\}$) iff $\chi_r^{2,J}(\overline{x}) \neq \chi_r^{2,J}(\overline{y})$.*

Proof. To appear in the full version. □

Corollary 3.5. *For two groups G,H of the same order and any $k \geq 1$, the following are equivalent:*

1. *The 2-ary k-pebble game does not distinguish two groups G,H*
2. *The multisets of stable colors on G^k and H^k are the same, that is, $\{\{\chi_\infty^{2,J}(\overline{x}) : \overline{x} \in G^k\}\} = \{\{\chi_\infty^{2,J}(\overline{y}) : \overline{y} \in H^k\}\}$*
3. *$\chi_\infty^{2,J}((1_G, 1_G, \ldots, 1_G)) = \chi_\infty^{2,J}((1_H, \ldots, 1_H))$.*

The analogous result holds in the $q = 1$ case, going back to [12].

Proof. To appear in the full version. □

Remark 3.6. For arbitrary relational structures with relations of arity $a+1$, the a-order pebble game may still be nontrivial, as pointed out in Hella [42, p. 6, just before §4]. Our coloring procedure generalizes in the following way to this more general setting, and the proof of the equivalence between the coloring procedure and Hella's pebble game is the same as the above, *mutatis mutandis*. The main change is that for an a-th order pebble game, instead of just considering a graph on edges of size 1 (when $i = j$) or 2 (when $i \neq j$), we consider an a'-uniform directed hypergraph, where each hyperedge consists of a list of a' vertices, for all $1 \leq a' \leq a$. This gives a coloring equivalent of the logical and game characterizations provided by Hella; this trifecta is partly why we feel it is justified to call this a "higher-arity Weisfeiler–Leman" coloring procedure.

We note that there has been some work on equivalences with specific binary and higher-arity quantifiers: see for instance, the invertible map game of Dawar & Holm [21] which generalizes rank logic, in which Spoiler can place multiple pebbles, but the bijections Duplicator selects must satisfy additional structure. Subsequently, Dawar & Vagnozzi [22] provided a generalization of Weisfeiler–Leman that further subsumes the invertible map game. We note that Dawar & Vagnozzi's "$WL_{k,r}$", although it looks superficially like our r-ary k-WL, is in fact quite different: in particular, their refinement step "flattens" a multiset of multisets into its multiset union, which loses information compared to our 2-ary (resp., r-ary) game; indeed, they show that their $\mathrm{WL}_{*,r}$ is equivalent to ordinary (1-ary) WL for any fixed r, whereas already 2-ary WL can solve GI. In general, the relationship between Hella's 2-ary game and the works of Dawar & Holm and Dawar & Vagnozzi remains open.

3.1 Equivalence between 2-ary (k,r)-WL Versions I and II

In this section we show that, up to additive constants in the number of pebbles and rounds, 2-ary WL Versions I and II are equivalent in their distinguishing power. For two different WL versions W, W', we write $W \preceq W'$ to mean that if W distinguishes two groups G and H, then so does W'.

Theorem 3.7. *Let $k \geq 2, r \geq 1$. We have that:*

$$(k,r)\text{-}WL_I^2 \preceq (k,r)\text{-}WL_{II}^2 \preceq (k+2,r+1)\text{-}WL_I^2.$$

Proof. To appear in the full version. □

4 Descriptive Complexity of Semisimple Groups

In this section, we show that the $(O(1),O(1))$-WL$_{II}^2$ pebble game can identify groups with no Abelian normal subgroups,[3] also known as semisimple groups. We begin with some preliminaries.

4.1 Preliminaries

Semisimple groups are motivated by the following characteristic filtration:

$$1 \leq \text{Rad}(G) \leq \text{Soc}^*(G) \leq \text{PKer}(G) \leq G,$$

which arises in the computational complexity community where it is known as the Babai–Beals filtration [5], as well as in the development of practical algorithms for computer algebra systems (c.f., [18]). We now explain the terms of this chain. Here, $\text{Rad}(G)$ is the *solvable radical*, which is the unique maximal solvable normal subgroup of G; recall that a group N is solvable if the sequence $N^{(0)} := N$, $N^{(i)} = [N^{(i-1)}, N^{(i-1)}]$ terminates in the trivial group after finitely many steps, and $[A,B]$ denotes the subgroup generated by $\{aba^{-1}b^{-1} : a \in A, b \in B\}$. The socle of a group, denoted $\text{Soc}(G)$, is the subgroup generated by all the minimal normal subgroups of G. $\text{Soc}^*(G)$ is the preimage of the socle $\text{Soc}(G/\text{Rad}(G))$ under the natural projection map $\pi : G \to G/\text{Rad}(G)$. To define PKer, we start by examining the action on $\text{Soc}(G/\text{Rad}(G)) \cong \text{Soc}^*(G)/\text{Rad}(G)$ that is induced by the action of G on $\text{Soc}^*(G)$ by conjugation. As $\text{Soc}^*(G)/\text{Rad}(G) \cong \text{Soc}(G/\text{Rad}(G))$ is the direct product of finite, non-Abelian simple groups $T_1, \ldots, T_k$, this action permutes the k simple factors, yielding a homomorphism $\varphi : G \to S_k$. The kernel of this action is denoted $\text{PKer}(G)$.

When $\text{Rad}(G)$ is trivial, G has no Abelian normal subgroups (and vice versa). We refer to such groups as *semisimple* (following [7, 8]) or trivial-Fitting (following [18]). As a semisimple group G has no Abelian normal subgroups, we have that $\text{Soc}(G)$ is the direct product of non-Abelian simple groups. As the conjugation action of G on $\text{Soc}(G)$ permutes the direct factors of $\text{Soc}(G)$, there exists a faithful permutation representation $\alpha : G \to G^* \leq \text{Aut}(\text{Soc}(G))$. G is determined by $\text{Soc}(G)$ and the action α. Let H be a semisimple group with the associated action $\beta : H \to \text{Aut}(\text{Soc}(H))$. We have that $G \cong H$ precisely if $\text{Soc}(G) \cong \text{Soc}(H)$ via an isomorphism that makes α equivalent to β in the sense introduced next.

We now introduce the notion of permutational isomorphism, which is our notion of equivalence for α and β. Let A and B be finite sets, and let $\pi : A \to B$ be a bijection. For $\sigma \in \text{Sym}(A)$, let $\sigma^\pi \in \text{Sym}(B)$

[3]In many places, we will use $O(1)$ for number of pebbles or rounds; we believe all of these can be replaced with particular numbers by a straightforward, if tedious, analysis of our proofs. However, since our focus is on the fact that these numbers are constant rather than on the exact values, we use the $O(1)$ notation.

be defined by $\sigma^\pi := \pi^{-1}\sigma\pi$. For a set $\Sigma \subseteq \mathrm{Sym}(A)$, denote $\Sigma^\pi := \{\sigma^\pi : \sigma \in \Sigma\}$. Let $K \leq \mathrm{Sym}(A)$ and $L \leq \mathrm{Sym}(B)$ be permutation groups. A bijection $\pi : A \to B$ is a *permutational isomorphism* $K \to L$ if $K^\pi = L$.

The following lemma, applied with $R = \mathrm{Soc}(G)$ and $S = \mathrm{Soc}(H)$, gives a precise characterization of semisimple groups in terms of the associated actions.

Lemma 4.1 ([7, Lemma 3.1], cf. [18, §3])**.** *Let G and H be groups, with $R \lhd G$ and $S \lhd H$ groups with trivial centralizers. Let $\alpha : G \to \mathrm{Aut}(R)$ and $\beta : H \to \mathrm{Aut}(S)$ be faithful permutation representations of G and H via the conjugation action on R and S, respectively. Let $f : R \to S$ be an isomorphism. Then f extends to an isomorphism $\hat{f} : G \to H$ if and only if f is a permutational isomorphism between $G^* = Im(\alpha)$ and $H^* = Im(\beta)$; and if so, $\hat{f} = \alpha f^* \beta^{-1}$, where $f^* : G^* \to H^*$ is the isomorphism induced by f.*

We also need the following standard group-theoretic lemmas. The first provides a key condition for identifying whether a non-Abelian simple group belongs to the socle. Namely, if $S_1 \cong S_2$ are non-Abelian simple groups where S_1 is in the socle and S_2 is not in the socle, then the normal closures of S_1 and S_2 are non-isomorphic. In particular, the normal closure of S_1 is a direct product of non-Abelian simple groups, while the normal closure of S_2 is not a direct product of non-Abelian simple groups. We will apply this condition later when S_1 is a simple direct factor of $\mathrm{Soc}(G)$; in which case, the normal closure of S_1 is of the form S_1^k.

Lemma 4.2 (c.f. [30, Lemma 6.5])**.** *Let G be a finite semisimple group. A subgroup $S \leq G$ is contained in $Soc(G)$ if and only if the normal closure of S is a direct product of nonabelian simple groups.*

Lemma 4.3 (c.f. [30, Lemma 6.6])**.** *Let $S_1, \ldots, S_k \leq G$ be nonabelian simple subgroups such that for all distinct $i, j \in [k]$ we have $[S_i, S_j] = 1$. Then $\langle S_1, \ldots, S_k \rangle = S_1 S_2 \cdots S_k = S_1 \times \cdots \times S_k$.*

4.2 Main Results

We show that the second Ehrenfeucht–Fraïssé game in Hella's hierarchy can identify both $\mathrm{Soc}(G)$ and the conjugation action when G is semisimple. We first show that this pebble game can identify whether a group is semisimple. Namely, if G is semisimple and H is not semisimple, then Spoiler can distinguish G from H.

Proposition 4.4. *Let G be a semisimple group of order n, and let H be an arbitrary group of order n. If H is not semisimple, then Spoiler can win in the $(4,2)$-WL_{II}^2 game.*

Proof. To appear in the full version. □

We now apply Lemma 4.2 to show that Duplicator must map the direct factors of $\mathrm{Soc}(G)$ to isomorphic direct factors of $\mathrm{Soc}(H)$.

Lemma 4.5. *Let G, H be finite groups of order n. Let $Fac(Soc(G))$ denote the set of simple direct factors of $Soc(G)$. Let $S \in Fac(Soc(G))$ be a non-Abelian simple group, with $S = \langle x, y \rangle$. If Duplicator selects a bijection $f \colon G \to H$ such that:*

(a) $S \not\cong \langle f(x), f(y) \rangle$, then Spoiler can win in the $(2,1)$-WL_{II}^2 game; or

(b) $f(S) \neq \langle f(x), f(y) \rangle$, then Spoiler can win in the $(4,2)$-WL_{II}^2 pebble game.

Note that the lemma does not require $f|_S \colon S \to f(S)$ to actually be an isomorphism, only that S and $f(S)$ are isomorphic.

Proof. To appear in the full version. □

Proposition 4.6. *Let G be a semisimple group of order n, and let H be an arbitrary group of order n. Let $f : G \to H$ be the bijection Duplicator selects. If there exists $S \in Fac(Soc(G))$ such that $f(S) \notin Fac(Soc(H))$ or $f(S) \not\cong S$, then Spoiler can win in the $(4,2)$-WL_{II}^2 pebble game.*

Proof. To appear in the full version. □

Lemma 4.7. *Let G,H be groups of order n, let S be a nonabelian simple group in $Fac(Soc(G))$. Let $f,f' : G \to H$ be two bijections selected by Duplicator at two different rounds. If $f(S) \cap f'(S) \neq 1$, then $f(S) = f'(S)$, or Spoiler can win in the $(4,2)$-WL_{II}^2 pebble game.*

Proof. By Prop. 4.6, both $f(S)$ and $f'(S)$ must be simple normal subgroups of $\mathrm{Soc}(H)$ (or Spoiler wins with 4 pebbles and 2 rounds). Since they intersect nontrivially, but distinct simple normal subgroups of $\mathrm{Soc}(H)$ intersect trivially, the two must be equal. □

We next introduce the notion of weight.

Definition 4.8. Let $\mathrm{Soc}(G) = S_1 \times \cdots \times S_k$ where each S_i is a simple normal subgroup of $\mathrm{Soc}(G)$. For any $s \in \mathrm{Soc}(G)$, write $s = s_1 s_2 \cdots s_k$ where each $s_i \in S_i$, and define the *weight* of s, denote $\mathrm{wt}(s)$, as the number of i's such that $s_i \neq 1$.

Note that the definition of weight is well-defined since the S_i are the unique subsets of $\mathrm{Soc}(G)$ that are simple normal subgroup of $\mathrm{Soc}(G)$, so the decomposition $s = s_1 s_2 \ldots s_k$ is unique up to the order of the factors. (This is essentially a particular instance of the "rank lemma" from [30], which intuitively states that WL detects in $O(\log n)$ rounds the set of elements for a given subgroup provided that it also identifies the generators. As we are now in the setting of 2-ary WL we give the full proof, which also has tighter bounds on the number of rounds.)

Lemma 4.9 (Weight Lemma)**.** *Let G,H be semisimple groups of order n. If Duplicator selects a bijection $f : G \to H$ that does not map $Soc(G)$ bijectively to $Soc(H)$, or does not preserve the weight of every element in $Soc(G)$, then Spoiler can win in the $(4,3)$-WL_{II}^2 game.*

Proof. To appear in the full version. □

Lemma 4.10. *Let G and H be semisimple groups with isomorphic socles. Let $S_1, S_2 \in Fac(Soc(G))$ be distinct. Let $f : G \to H$ be the bijection that Duplicator selects. If there exist $x_i \in S_i$ such that $f(x_1x_2) \neq f(x_1)f(x_2)$, then Spoiler can win in the $(4,3)$-WL_{II}^2 pebble game.*

Proof. By Lem. 4.9, we may assume that $\mathrm{wt}(s) = \mathrm{wt}(f(s))$ for all $s \in \mathrm{Soc}(G)$; otherwise, Spoiler wins with at most 4 pebbles and 3 rounds. As $f(x_1x_2)$ has weight 2, $f(x_1x_2)$ belongs to the direct product of two simple factors in $\mathrm{Fac}(\mathrm{Soc}(H))$, so it can be written $f(x_1x_2) = y_1y_2$ with each y_i in distinct simple factors in $\mathrm{Fac}(\mathrm{Soc}(H))$. Without loss of generality suppose that $y_1 \neq f(x_1)$. Spoiler pebbles $(x_1, x_1x_2) \mapsto (f(x_1), f(x_1x_2))$. Now $\mathrm{wt}(x_1^{-1} \cdot x_1x_2) = 1$, while $\mathrm{wt}(f(x_1)^{-1} \cdot f(x_1x_2)) \geq 2$. (Note that we cannot quite yet directly apply Lem. 4.9, because we have not yet identified a single element x such that $\mathrm{wt}(x) \neq \mathrm{wt}(f(x))$.)

On the next round, Duplicator selects another bijection f'. Spoiler now pebbles $x_2 \mapsto f'(x_2)$. Because $\mathrm{wt}(x_1^{-1} \cdot x_1x_2) = 1$ but $\mathrm{wt}(f(x_1)^{-1} f(x_1x_2)) \geq 2$, and f' preserves weight by Lem. 4.9, we have $f'(x_2) \neq f'(x_1)^{-1} f'(x_1x_2)$. Thus, the pebbled map $(x_1, x_2, x_1x_2) \mapsto (f'(x_1), f(x_2), f(x_1x_2))$ does not extend to an isomorphism, and so Spoiler wins with 3 pebbles and 2 rounds. □

Recall that if G is semisimple, then $G \leq \text{Aut}(\text{Soc}(G))$. Now each minimal normal subgroup $N \trianglelefteq G$ is of the form $N = S^k$, where S is a non-Abelian simple group. So $\text{Aut}(N) = \text{Aut}(S) \wr \text{Sym}(k)$. In particular,

$$G \leq \prod_{\substack{N \trianglelefteq G \\ N \text{ is minimal normal}}} \text{Aut}(N).$$

So if $g \in G$, then the conjugation action of g on $\text{Soc}(G)$ acts by (i) automorphism on each simple direct factor of $\text{Soc}(G)$, and (ii) by permuting the direct factors of $\text{Soc}(G)$. Provided generators of the direct factors of the socle are pebbled, Spoiler can detect inconsistencies of the automorphism action. However, doing so directly would be too expensive as there could be $\Theta(\log |G|)$ generators, so we employ a more subtle approach with a similar outcome. By Lem. 4.9, Duplicator must select bijections $f : G \to H$ that preserve weight. That is, if $s \in \text{Soc}(G)$, then $\text{wt}(s) = \text{wt}(f(s))$. We use Lem. 4.9 in tandem with the fact that the direct factors of the socle commute to effectively pebble the set of all the generators at once. Namely, suppose that $\text{Fac}(\text{Soc}(G)) = \{S_1, \ldots, S_k\}$, where $S_i = \langle x_i, y_i \rangle$. Let $x := x_1 \cdots x_k$ and $y := y_1 \cdots y_k$. We will show that it suffices for Spoiler to pebble (x, y) rather than individually pebbling generators for each S_i (this will still allow the factors to be permuted, but that is all).

Lemma 4.11. *Let G and H be semisimple groups with isomorphic socles, and write $Fac(Soc(G)) = \{S_1, \ldots, S_m\}$, with $S_i = \langle x_i, y_i \rangle$. Let $f : G \to H$ be the bijection that Duplicator selects, and suppose that (i) for all i, $f(S_i) \cong S_i$ (though $f|_{S_i}$ need not be an isomorphism) and $f(S_i) \in Fac(Soc(H))$, (ii) for every $s \in Soc(G)$, $wt(s) = wt(f(s))$, and (iii) for all i, $f(S_i) = \langle f(x), f(y) \rangle$.*

Now suppose that Spoiler pebbles $(x_1 \cdots x_m, y_1 \cdots y_m) \mapsto (f(x_1 \cdots x_m), f(y_1 \cdots y_m))$. As f preserves weight, we may write $f(x_1 \cdots x_m) = h_1 \cdots h_m$ and $f(y_1 \cdots y_m) = z_1 \cdots z_m$ with $h_i, z_i \in f(S_i)$ for all i.

Let $f' : G \to H$ be the bijection that Duplicator selects at any subsequent round in which the pebble used above has not moved. If any of the following hold, then Spoiler can win in the WL_{II}^2 pebble game with 5 additional pebbles and 5 additional rounds:

(a) f' does not satisfy conditions (i)–(iii),

(b) there exists an $i \in [m]$ such that $f'(x_i) \notin \{h_1, \ldots, h_m\}$ or $f'(y_i) \notin \{z_1, \ldots, z_m\}$

(c) $f'|_{S_i}$ is not an isomorphism

(d) there exists $g \in G$ and $i \in [m]$ such that $gS_ig^{-1} = S_i$ and for some $x \in S_i$, the following holds: $f'(gxg^{-1}) \neq f'(g)f'(x)f'(g)^{-1}$.

Proof. To appear in the full version. □

Lem. 4.11 provides enough to establish that Spoiler can force Duplicator to select at each round a bijection that restricts to an isomorphism on the socles.

Proposition 4.12. *(Same assumptions as Lem. 4.11.) Let G and H be semisimple groups with isomorphic socles, with $Fac(Soc(G)) = \{S_1, \ldots, S_m\}$, with $S_i = \langle x_i, y_i \rangle$. Let $f_0 : G \to H$ be the bijection that Duplicator selects, and suppose that (i) for all i, $f_0(S_i) \cong S_i$ (though $f_0|_{S_i}$ need not be an isomorphism) and $f_0(S_i) \in Fac(Soc(H))$, (ii) for every $s \in Soc(G)$, $wt(s) = wt(f_0(s))$, and (iii) for all i, $f_0(S_i) = \langle f_0(x), f_0(y) \rangle$. Now suppose that Spoiler pebbles $(x_1 \cdots x_m, y_1 \cdots y_m) \mapsto (f_0(x_1 \cdots x_m), f_0(y_1 \cdots y_m))$.*

Let $f' : G \to H$ be the bijection that Duplicator selects at any subsequent round in which the pebbles used above have not moved. Then $f'|_{Soc(G)} : Soc(G) \to Soc(H)$ must be an isomorphism, or Spoiler can win in 4 more rounds using at most 6 more pebbles (for a total of 7 pebbles and 5 rounds) in the WL_{II}^2 pebble game.

Proof. To appear in the full version. □

Remark 4.13. Brachter & Schweitzer [13, Lemma 5.22] previously showed that (1-ary) Weisfeiler–Leman can decide whether two groups have isomorphic socles. However, their results did not solve the search problem; that is, they did not show Duplicator must select bijections that restrict to an isomorphism on the socle even in the case for semisimple groups. This contrasts with Lem. 4.12, where we show that 2-ary WL effectively solves the search problem. This is an important ingredient in our proof that the $(7, O(1))$-WL_{II}^2 pebble game solves isomorphism for semisimple groups.

We obtain as a corollary of Lem. 4.11 and Lem. 4.12 that if G and H are semisimple, then Duplicator must select bijections that restrict to isomorphisms of $\text{PKer}(G)$ and $\text{PKer}(H)$.

Corollary 4.14. *Let G and H be semisimple groups of order n. Let $Fac(Soc(G)) := \{S_1, \ldots, S_m\}$, and suppose that $S_i = \langle x_i, y_i \rangle$. Let $x := x_1 \cdots x_m$ and $y := y_1 \cdots y_m$. and Let $f : G \to H$ be the bijection that Duplicator selects. Spoiler begins by pebbling $(x, y) \mapsto (f(x), f(y))$. Let $f' : G \to H$ be the bijection that Duplicator selects at the next round. If $f'|_{\text{PKer}(G)} : \text{PKer}(G) \to \text{PKer}(H)$ is not an isomorphism, then Spoiler can win with 5 additional pebbles and 5 additional rounds in the WL_{II}^2 pebble game.*

Proof. To appear in the full version. □

We now show that if G and H are not permutationally equivalent, then Spoiler can win.

Lemma 4.15. *(Same assumptions as Lem. 4.11.) Let G and H be semisimple groups with isomorphic socles, with $Fac(Soc(G)) = \{S_1, \ldots, S_m\}$, with $S_i = \langle x_i, y_i \rangle$. Let $f_0 : G \to H$ be the bijection that Duplicator selects, and suppose that (i) for all i, $f_0(S_i) \cong S_i$ (though $f_0|_{S_i}$ need not be an isomorphism) and $f_0(S_i) \in Fac(Soc(H))$, (ii) for every $s \in Soc(G)$, $wt(s) = wt(f_0(s))$, and (iii) for all i, $f_0(S_i) = \langle f_0(x), f_0(y) \rangle$. Now suppose that Spoiler pebbles $(x_1 \cdots x_m, y_1 \cdots y_m) \mapsto (f_0(x_1 \cdots x_m), f_0(y_1 \cdots y_m))$.*

Let $f' : G \to H$ be the bijection that Duplicator selects at the next round. Suppose that there exist $g \in G$ and $i \in [m]$ such that $f'(gS_ig^{-1}) = f'(S_j)$, but $f'(g)f'(S_i)f'(g)^{-1} = f'(S_k)$ for some $k \neq j$. Then Spoiler can win with 4 additional pebbles and 4 additional rounds in the WL_{II}^2 pebble game.

Proof. To appear in the full version. □

Theorem 4.16. *Let G be a semisimple group and H an arbitrary group of order n, not isomorphic to G. Then Spoiler has a winning strategy in the $(9, O(1))$-WL_{II}^2 pebble game.*

Proof. If H is not semisimple, then by Prop. 4.4, Spoiler wins with 4 pebbles and 2 rounds. So we now suppose H is semisimple.

Let $\text{Fac}(\text{Soc}(G)) = \{S_1, \ldots, S_k\}$, and let x_i, y_i be generators of S_i for each i. Let f be the bijection chosen by Duplicator. Spoiler pebbles $(x_1x_2 \cdots x_k, y_1y_2, \ldots, y_k) \mapsto (f(x_1 \cdots x_k), f(y_1 \cdots y_k))$. On subsequent rounds, we thus have satisfied the hypotheses of Lem. 4.11 and Prop. 4.12. Spoiler will never move this pebble, and thus all subsequent bijections chosen by Duplicator must restrict to isomorphisms on the socle (or Spoiler wins with at most 7 pebbles and $O(1)$ rounds).

Recall from Lem. 4.1 that $G \cong H$ iff there is an isomorphism $\mu : \text{Soc}(G) \to \text{Soc}(H)$ that induces a permutational isomorphism $\mu^* : G^* \to H^*$. Thus, since $G \not\cong H$, there must be some $g \in G$ and $s \in \text{Soc}(G)$ such that $f(gsg^{-1}) \neq f(g)f(s)f(g)^{-1}$. Write $s = s_1 \cdots s_k$ with each $s_i \in S_i$ (not necessarily nontrivial).

We claim that there exists some i such that $f(gs_ig^{-1}) \neq f(g)f(s_i)f(g)^{-1}$. For suppose not, then we have

$$\begin{aligned}
f(gsg^{-1}) &= f(gs_1g^{-1}gs_2g^{-1}\cdots gs_kg^{-1}) \\
&= f(gs_1g^{-1})f(gs_2g^{-1})\cdots f(gs_kg^{-1}) \\
&= f(g)f(s_1)f(g)^{-1}f(g)f(s_2)f(g)^{-1}\cdots f(g)f(s_k)f(g)^{-1} \\
&= f(g)f(s_1\cdots s_k)f(g)^{-1} = f(g)f(s)f(g)^{-1},
\end{aligned}$$

a contradiction. For simplicity of notation, without loss of generality we may assume $i = 1$, so we now have $f(gs_1g)^{-1} \neq f(g)f(s_1)f(g)^{-1}$.

We break the argument into cases:

1. If $gs_1g^{-1} \in S_1$, then we have $gS_1g^{-1} = S_1$ (any two distinct simple normal factors of the socle intersect trivially), we have by Lem. 4.11 (d) that Spoiler can win with at most 5 additional pebbles (for a total of 7 pebbles) and 5 additional rounds (for a total of 6 rounds).
2. If $gs_1g^{-1} \in S_j$ for $j \neq 1$ and $f(g)f(s_1)f(g)^{-1} \notin f(S_j)$, we have by Lem. 4.15 that Spoiler can win with at most 4 additional pebbles (for a total of 6 pebbles) and 4 additional rounds (for a total of 5 rounds).
3. Suppose now that $gs_1g^{-1} \in S_j$ for some $j \neq 1$ and $f(g)f(s_1)f(g)^{-1} \in f(S_j)$. Spoiler begins by pebbling $(g, gs_1g^{-1}) \mapsto (f(g), f(gs_1g^{-1}))$. Let $f' : G \to H$ be the bijection that Duplicator selects at the next round. As $gs_1g^{-1} \in S_j$ is pebbled, we have that $f'(S_j) = f(S_j)$ by Lem. 4.7 (or Spoiler wins with 4 additional pebbles and 2 additional rounds). Now by assumption, $gS_1g^{-1} = S_j$ and $f(g)f(S_1)f(g)^{-1} = f(S_j)$. So as $g \mapsto f(g)$ is pebbled, we claim that we may assume $f'(S_1) = f(S_1)$. For suppose not; then we have $g^{-1}S_jg = S_1$ but $f'(g)^{-1}f'(S_j)f'(g) = f(g)^{-1}f(S_j)f(g) = f(S_1) \neq f'(S_1)$. But then Spoiler can with win with 4 additional pebbles (for a total of 8 pebbles) and 4 additional rounds (for a total of 7 rounds) by Lem. 4.15. Thus we have $f'(S_1) = f(S_1)$.

   In particular, we have that $f'(x_1) = f(x_1)$ and $f'(y_1) = f(y_1)$, by the same argument as in the proof of Lem. 4.11 (c). As $S_1 = \langle x_1, y_1 \rangle$, we have that $f'(s_1) = f(s_1)$, since they are both isomorphisms on the socle by Prop. 4.12. Spoiler now pebbles $(x_1, y_1) \mapsto (f'(x_1), f'(y_1))$. As the pebbled map $(g, x_1, y_1, gs_1g^{-1}) \mapsto (f(g), f'(x_1), f'(y_1), f'(gs_1g^{-1}))$ does not extend to an isomorphism, Spoiler wins. In this case, Spoiler used at most 8 pebbles and 7 rounds.

Note that the ninth pebble is the one we pick up prior to checking the winning condition. □

5 Conclusion

We exhibited a novel Weisfeiler–Leman algorithm that provides an algorithmic characterization of the second Ehrenfeucht–Fraïssé game in Hella's [41, 42] hierarchy. We also showed that this Ehrenfeucht–Fraïssé game can identify groups without Abelian normal subgroups using $O(1)$ pebbles and $O(1)$ rounds. In particular, within the first few rounds, Spoiler can force Duplicator to select an isomorphism at each subsequent round. This effectively solves the search problem in the pebble game characterization.

Our work leaves several directions for further research.

Question 5.1. Can the constant-dimensional 2-ary Wesifeiler–Leman algorithm be implemented in time $n^{o(\log n)}$?

Question 5.2. What is the (1-ary) Weisfeiler–Leman dimension of groups without Abelian normal subgroups?

Question 5.3. Show that the second Ehrenfeucht–Fraïssé game in Hella's hierarchy can identify coprime extensions of the form $H \ltimes N$ with both H, N Abelian (the analogue of [62]). More generally, an analogue of Babai–Qiao [9] would be to show that when $|H|, |N|$ are coprime and N is Abelian, that Spoiler can distinguish $H \ltimes N$ from any non-isomorphic group using a constant number of pebbles that is no more than that which is required to identify H (or the maximum of that of H and a constant independent of N, H).

Question 5.4. Let $p > 2$ be prime, and let G be a p-group with bounded genus. Show that in the second Ehrenfeucht–Fraïssé game in Hella's hierarchy, Spoiler has a winning strategy using a constant number of pebbles. This is a descriptive complexity analogue of [15, 46]. It would even be of interest to start with the case where G has bounded genus over a field extension $K/\mathbb{F}_p$ of bounded degree.

In the setting of groups, Hella's hierarchy collapses to some $q \leq 3$, since 3-ary WL can identify all ternary relational structures, including groups. It remains open to determine whether this hierarchy collapses further to either $q = 1$ or $q = 2$. Even if it does not collapse, it would also be of interest to determine whether the 1-ary and 2-ary games are equivalent. Algorithmically, this is equivalent to determining whether 1-ary and 2-ary WL are have the same distinguishing power.

Question 5.5. Does there exist an infinite family of non-isomorphic pairs of groups $\{(G_n, H_n)\}$ for which Spoiler requires $\omega(1)$ pebbles to distinguish G_n from H_n? We ask this question for the Ehrenfeucht–Fraïssé games at both the first and second levels of Hella's hierarchy.

Recall that the game at the first level of Hella's hierarchy is equivalent to Weisfeiler–Leman [17, 41, 42], and so a lower bound against either of these games provides a lower bound against Weisfeiler–Leman. More generally, it would also be of interest to investigate Hella's hierarchy on higher arity structures. For a q-ary relational structure, the q-ary pebble game suffices to decide isomorphism. Are there interesting, natural classes of higher arity structures for which Hella's hierarchy collapses further to some level $q' < q$?

Acknowledgment

The authors would like to thank Pascal Schweitzer and the anonymous referees for feedback on earlier versions of some of the results in this paper, as well as discussions more directly relevant to the paper. JAG would like to thank Martin Grohe for helpful correspondence about WL versus "oblivious WL." JAG was partially supported by NSF award DMS-1750319 and NSF CAREER award CCF-2047756 and during this work. ML was partially supported by J. Grochow startup funds, NSF award CISE-2047756, and a Summer Research Fellowship through the Department of Computer Science at the University of Colorado Boulder.

References

[1] Miklos Ajtai & Ronald Fagin (1990): *Reachability is harder for directed than for undirected finite graphs*. Journal of Symbolic Logic 55(1), p. 113–150, doi:10.2307/2274958.

[2] Sanjeev Arora & Ronald Fagin (1997): *On winning strategies in Ehrenfeucht–Fraïssé games*. Theoretical Computer Science 174(1), pp. 97–121, doi:10.1016/S0304-3975(96)00015-1.

[3] V. Arvind & Piyush P. Kurur (2006): *Graph Isomorphism is in SPP*. Information and Computation 204(5), pp. 835–852, doi:10.1016/j.ic.2006.02.002.

[4] László Babai (2016): *Graph isomorphism in quasipolynomial time [extended abstract]*. In: *STOC'16—Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, ACM, New York, pp. 684–697, doi:10.1145/2897518.2897542. Preprint of full version at arXiv:1512.03547v2 [cs.DS].

[5] László Babai & Robert Beals (1999): *A polynomial-time theory of black box groups I*. In: *Groups St Andrews 1997 in Bath, I*, doi:10.1017/CBO9781107360228.004.

[6] László Babai, Robert Beals & Ákos Seress (2009): *Polynomial-Time Theory of Matrix Groups*. In: *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, Association for Computing Machinery, p. 55–64, doi:10.1145/1536414.1536425.

[7] László Babai, Paolo Codenotti, Joshua A. Grochow & Youming Qiao (2011): *Code equivalence and group isomorphism*. In: *Proceedings of the Twenty-Second Annual ACM–SIAM Symposium on Discrete Algorithms (SODA11)*, SIAM, Philadelphia, PA, pp. 1395–1408, doi:10.1137/1.9781611973082.107.

[8] László Babai, Paolo Codenotti & Youming Qiao (2012): *Polynomial-Time Isomorphism Test for Groups with No Abelian Normal Subgroups - (Extended Abstract)*. In: *International Colloquium on Automata, Languages, and Programming (ICALP)*, pp. 51–62, doi:10.1007/978-3-642-31594-7_5.

[9] László Babai & Youming Qiao (2012): *Polynomial-time Isomorphism Test for Groups with Abelian Sylow Towers*. In: *29th STACS*, Springer LNCS 6651, pp. 453 – 464, doi:10.4230/LIPIcs.STACS.2012.453.

[10] Hans Ulrich Besche & Bettina Eick (1999): *Construction of finite groups*. *J. Symb. Comput.* 27(4), pp. 387–404, doi:10.1006/jsco.1998.0258.

[11] Hans Ulrich Besche, Bettina Eick & E.A. O'Brien (2002): *A Millennium Project: Constructing Small Groups*. *Intern. J. Alg. and Comput* 12, pp. 623–644, doi:10.1142/S0218196702001115.

[12] Jendrik Brachter & Pascal Schweitzer (2020): *On the Weisfeiler–Leman Dimension of Finite Groups*. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi & Dale Miller, editors: *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, ACM, pp. 287–300, doi:10.1145/3373718.3394786.

[13] Jendrik Brachter & Pascal Schweitzer (2021): *A Systematic Study of Isomorphism Invariants of Finite Groups via the Weisfeiler–Leman Dimension*. arXiv:2111.11908 [math.GR].

[14] Peter A. Brooksbank, Joshua A. Grochow, Yinan Li, Youming Qiao & James B. Wilson (2019): *Incorporating Weisfeiler–Leman into algorithms for group isomorphism*. arXiv:1905.02518 [cs.CC].

[15] Peter A. Brooksbank, Joshua Maglione & James B. Wilson (2017): *A fast isomorphism test for groups whose Lie algebra has genus 2*. *Journal of Algebra* 473, pp. 545–590, doi:10.1016/j.jalgebra.2016.12.007.

[16] Harry Buhrman & Steven Homer (1992): *Superpolynomial Circuits, Almost Sparse Oracles and the Exponential Hierarchy*. In R. K. Shyamasundar, editor: *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings*, *Lecture Notes in Computer Science* 652, Springer, pp. 116–127, doi:10.1007/3-540-56287-7_99.

[17] Jin-Yi Cai, Martin Fürer & Neil Immerman (1992): *An optimal lower bound on the number of variables for graph identification*. *Combinatorica* 12(4), pp. 389–410, doi:10.1007/BF01305232. Originally appeared in SFCS '89.

[18] John J. Cannon & Derek F. Holt (2003): *Automorphism group computation and isomorphism testing in finite groups*. *J. Symb. Comput.* 35, pp. 241–267, doi:10.1016/S0747-7171(02)00133-5.

[19] Arkadev Chattopadhyay, Jacobo Torán & Fabian Wagner (2013): *Graph isomorphism is not* AC^0*-reducible to group isomorphism*. *ACM Trans. Comput. Theory* 5(4), pp. Art. 13, 13, doi:10.1145/2540088. Preliminary version appeared in FSTTCS '10; ECCC Tech. Report TR10-117.

[20] Bireswar Das & Shivdutt Sharma (2019): *Nearly Linear Time Isomorphism Algorithms for Some Nonabelian Group Classes*. In René van Bevern & Gregory Kucherov, editors: *Computer Science – Theory and Applications*, Springer International Publishing, Cham, pp. 80–92, doi:10.1007/s00224-020-10010-z.

[21] Anuj Dawar & Bjarki Holm (2012): *Pebble Games with Algebraic Rules*. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts & Roger Wattenhofer, editors: *Automata, Languages, and Programming*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 251–262, doi:10.1007/978-3-642-31585-5_25.

[22] Anuj Dawar & Danny Vagnozzi (2020): *Generalizations of k-dimensional Weisfeiler–Leman stabilization*. *Moscow Journal of Combinatorics and Number Theory* 9, pp. 229–252, doi:10.2140/moscow.2020.9.229.

[23] Heiko Dietrich & James B. Wilson (2022): *Polynomial-time isomorphism testing for groups of most finite orders*, doi:10.1109/FOCS52979.2021.00053.

[24] A. Ehrenfeucht (1960/61): *An application of games to the completeness problem for formalized theories*. *Fund. Math.* 49, pp. 129–141, doi:10.4064/fm-49-2-129-141.

[25] Bettina Eick, C. R. Leedham-Green & E. A. O'Brien (2002): *Constructing automorphism groups of p-groups*. *Comm. Algebra* 30(5), pp. 2271–2295, doi:10.1081/AGB-120003468.

[26] Jörg Flum & Martin Grohe (2000): *On Fixed-Point Logic with Counting*. *The Journal of Symbolic Logic* 65(2), pp. 777–787, doi:10.2307/2586569.

[27] Roland Fraïssé (1954): *Sur quelques classifications des systèmes de relations*. *Publ. Sci. Univ. Alger. Sér. A* 1, pp. 35–182 (1955).

[28] Walid Gomaa (2010): *Descriptive Complexity of Finite Abelian Groups*. *IJAC* 20, pp. 1087–1116, doi:10.1142/S0218196710006047.

[29] Joshua A. Grochow & Michael Levet (2022): *On the Descriptive Complexity of Groups without Abelian Normal Subgroups*. arXiv:2209.13725.

[30] Joshua A. Grochow & Michael Levet (2022): *On the parallel complexity of Group Isomorphism and canonization via Weisfeiler–Leman*. arXiv:2112.11487 [cs.DS].

[31] Joshua A. Grochow & Youming Qiao (2015): *Polynomial-Time Isomorphism Test of Groups that are Tame Extensions - (Extended Abstract)*. In: *Algorithms and Computation - 26th International Symposium, ISAAC 2015, Nagoya, Japan, December 9-11, 2015, Proceedings*, pp. 578–589, doi:10.1007/978-3-662-48971-0_49.

[32] Martin Grohe (2017): *Descriptive complexity, canonisation, and definable graph structure theory*. *Lecture Notes in Logic* 47, Association for Symbolic Logic, Ithaca, NY; Cambridge University Press, Cambridge, doi:10.1017/9781139028868.

[33] Martin Grohe (2021): *The logic of graph neural networks*. In: *LICS '21: Proceedings of the 36th Annual ACM/IEEE Symposium on Logic in Computer Science*, doi:10.1109/LICS52264.2021.9470677. Preprint arXiv:2104.14624 [cs.LG].

[34] Martin Grohe & Sandra Kiefer (2019): *A Linear Upper Bound on the Weisfeiler–Leman Dimension of Graphs of Bounded Genus*. arXiv:1904.07216.

[35] Martin Grohe & Sandra Kiefer (2021): *Logarithmic Weisfeiler–Leman Identifies All Planar Graphs*. arXiv:2106.16218.

[36] Martin Grohe & Daniel Neuen (2019): *Canonisation and Definability for Graphs of Bounded Rank Width*. arXiv:1901.10330.

[37] Martin Grohe & Martin Otto (2015): *Pebble Games and linear equations*. *J. Symb. Log.* 80(3), pp. 797–844, doi:10.1017/jsl.2015.28.

[38] Martin Grohe & Oleg Verbitsky (2006): *Testing Graph Isomorphism in Parallel by Playing a Game*. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone & Ingo Wegener, editors: *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part I, Lecture Notes in Computer Science* 4051, Springer, pp. 3–14, doi:10.1007/11786986_2.

[39] Xiaoyu He & Youming Qiao (2021): *On the Baer–Lovász–Tutte construction of groups from graphs: Isomorphism types and homomorphism notions*. *Eur. J. Combin.* 98, p. 103404, doi:10.1016/j.ejc.2021.103404.

[40] Hermann Heineken & Hans Liebeck (1974): *The occurrence of finite groups in the automorphism group of nilpotent groups of class* 2. Arch. Math. (Basel) 25, pp. 8–16, doi:10.1007/BF01238631.

[41] Lauri Hella (1989): *Definability hierarchies of generalized quantifiers*. Annals of Pure and Applied Logic 43(3), pp. 235 – 271, doi:10.1016/0168-0072(89)90070-5.

[42] Lauri Hella (1996): *Logical Hierarchies in PTIME*. Information and Computation 129(1), pp. 1–19, doi:10.1006/inco.1996.0070.

[43] Neil Immerman (1986): *Relational Queries Computable in Polynomial Time*. Inf. Control. 68(1-3), pp. 86–104, doi:10.1016/S0019-9958(86)80029-8.

[44] Neil Immerman & Eric Lander (1990): *Describing Graphs: A First-Order Approach to Graph Canonization*. In Alan L. Selman, editor: Complexity Theory Retrospective: In Honor of Juris Hartmanis on the Occasion of His Sixtieth Birthday, July 5, 1988, Springer New York, New York, NY, pp. 59–81, doi:10.1007/978-1-4612-4478-3_5.

[45] Russell Impagliazzo, Ramamohan Paturi & Francis Zane (2001): *Which Problems Have Strongly Exponential Complexity?* Journal of Computer and System Sciences 63(4), pp. 512–530, doi:10.1006/jcss.2001.1774.

[46] Gábor Ivanyos & Youming Qiao (2019): *Algorithms Based on *-Algebras, and Their Applications to Isomorphism of Polynomials with One Secret, Group Isomorphism, and Polynomial Identity Testing*. SIAM J. Comput. 48(3), pp. 926–963, doi:10.1137/18M1165682.

[47] T. Kavitha (2007): *Linear time algorithms for Abelian group isomorphism and related problems*. Journal of Computer and System Sciences 73(6), pp. 986 – 996, doi:10.1016/j.jcss.2007.03.013.

[48] Neeraj Kayal & Timur Nezhmetdinov (2009): *Factoring Groups Efficiently*. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas & Wolfgang Thomas, editors: Automata, Languages and Programming, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 585–596, doi:10.1007/978-3-642-02927-1_49.

[49] Sandra Kiefer & Brendan D. McKay (2020): *The Iteration Number of Colour Refinement*. In Artur Czumaj, Anuj Dawar & Emanuela Merelli, editors: 47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference), LIPIcs 168, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 73:1–73:19, doi:10.4230/LIPIcs.ICALP.2020.73.

[50] Sandra Kiefer, Ilia Ponomarenko & Pascal Schweitzer (2019): *The Weisfeiler–Leman Dimension of Planar Graphs Is at Most 3*. J. ACM 66(6), doi:10.1145/3333003.

[51] Sandra Kiefer, Pascal Schweitzer & Erkal Selman (2022): *Graphs Identified by Logics with Counting*. ACM Trans. Comput. Log. 23(1), pp. 1:1–1:31, doi:10.1145/3417515.

[52] Johannes Köbler, Uwe Schöning & Jacobo Torán (1992): *Graph Isomorphism is Low for PP*. Comput. Complex. 2, pp. 301–330, doi:10.1007/BF01200427.

[53] Richard E. Ladner (1975): *On the Structure of Polynomial Time Reducibility*. J. ACM 22(1), p. 155–171, doi:10.1145/321864.321877.

[54] François Le Gall (2009): *Efficient Isomorphism Testing for a Class of Group Extensions*. In: Proc. 26th STACS, pp. 625–636, doi:10.4230/LIPIcs.STACS.2009.1830.

[55] François Le Gall & David J. Rosenbaum (2016): *On the Group and Color Isomorphism Problems*. arXiv:1609.08253 [cs.CC].

[56] Mark L. Lewis & James B. Wilson (2012): *Isomorphism in expanding families of indistinguishable groups*. Groups - Complexity - Cryptology 4(1), pp. 73–110, doi:10.1515/gcc-2012-0008.

[57] P. Lindstrom (1966): *First Order Predicate Logic with Generalized Quantifiers*. Theoria 32(3), pp. 186–195, doi:10.1111/j.1755-2567.1966.tb00600.x.

[58] Alan H. Mekler (1981): *Stability of Nilpotent Groups of Class 2 and Prime Exponent*. The Journal of Symbolic Logic 46(4), pp. 781–788, doi:10.2307/2273227. Available at http://www.jstor.org/stable/2273227.

[59] Gary L. Miller (1978): *On the $n^{\log n}$ Isomorphism Technique (A Preliminary Report).* In: *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, Association for Computing Machinery, New York, NY, USA, pp. 51–58, doi:10.1145/800133.804331.

[60] Andrzej Mostowski (1957): *On a generalization of quantifiers. Fundamenta Mathematicae* 44(1), pp. 12–36, doi:10.4064/fm-44-1-12-36. Available at http://eudml.org/doc/213418.

[61] André Nies & Katrin Tent (2017): *Describing finite groups by short first-order sentences. Israel J. Math.* 221(1), pp. 85–115, doi:10.1007/s11856-017-1563-2.

[62] Youming Qiao, Jayalal M. N. Sarma & Bangsheng Tang (2011): *On Isomorphism Testing of Groups with Normal Hall Subgroups.* In: *Proc. 28th STACS*, pp. 567–578, doi:10.4230/LIPIcs.STACS.2011.567.

[63] David J. Rosenbaum (2013): *Bidirectional Collision Detection and Faster Deterministic Isomorphism Testing.* arXiv:1304.3935 [cs.DS].

[64] Benjamin Rossman (2009): *Ehrenfeucht–Fraïssé Games on Random Structures.* In Hiroakira Ono, Makoto Kanazawa & Ruy J. G. B. de Queiroz, editors: *Logic, Language, Information and Computation, 16th International Workshop, WoLLIC 2009, Tokyo, Japan, June 21-24, 2009. Proceedings, Lecture Notes in Computer Science* 5514, Springer, pp. 350–364, doi:10.1007/978-3-642-02261-6_28.

[65] C. Savage (1980): *An $O(n^2)$ Algorithm for Abelian Group Isomorphism.* Technical Report, North Carolina State University.

[66] Uwe Schöning (1988): *Graph isomorphism is in the low hierarchy. Journal of Computer and System Sciences* 37(3), pp. 312 – 323, doi:10.1016/0022-0000(88)90010-4.

[67] Moshe Y. Vardi (1982): *The Complexity of Relational Query Languages (Extended Abstract).* In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard & Lawrence H. Landweber, editors: *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, ACM, pp. 137–146, doi:10.1145/800070.802186.

[68] Narayan Vikas (1996): *An $O(n)$ Algorithm for Abelian p-Group Isomorphism and an $O(n \log n)$ Algorithm for Abelian Group Isomorphism. Journal of Computer and System Sciences* 53(1), pp. 1–9, doi:10.1006/jcss.1996.0045.

[69] James B. Wilson (2012): *Existence, algorithms, and asymptotics of direct product decompositions, I. Groups - Complexity - Cryptology* 4(1), doi:10.1515/gcc-2012-0007.

[70] James B. Wilson (2019): *The Threshold for Subgroup Profiles to Agree is Logarithmic. Theory of Computing* 15(19), pp. 1–25, doi:10.4086/toc.2019.v015a019.

[71] V. N. Zemlyachenko, N. M. Korneenko & R. I. Tyshkevich (1985): *Graph isomorphism problem. J. Soviet Math.* 29(4), pp. 1426–1481, doi:10.1007/BF02104746.

An Objective Improvement Approach to Solving Discounted Payoff Games

Daniele Dell'Erba
University of Liverpool,
United Kingdom
daniele.dell-erba@liverpool.ac.uk

Arthur Dumas
ENS Rennes,
France
arthur.dumas@ens.rennes.fr

Sven Schewe
University of Liverpool,
United Kingdom
sven.schewe@liverpool.ac.uk

While discounted payoff games and classic games that reduce to them, like parity and mean-payoff games, are symmetric, their solutions are not. We have taken a fresh view on the constraints that optimal solutions need to satisfy, and devised a novel way to converge to them, which is entirely symmetric. It also challenges the gospel that methods for solving payoff games are either based on strategy improvement or on value iteration.

1 Introduction

We study turn-based zero sum games played between two players on directed graphs. The two players take turns to move a token along the vertices of finite labelled graph with the goal to optimise their adversarial objectives.

Various classes of graph games are characterised by the objective of the players, for instance in *parity games* the objective is to optimise the parity of the dominating colour occurring infinitely often, while in *discounted and mean-payoff games* the objective of the players is to minimise resp. maximise the discounted and limit-average sum of the colours.

Solving graph games is the central and most expensive step in many model checking [22, 12, 36, 1, 2, 31], satisfiability checking [36, 22, 34], and synthesis [28, 32] algorithms. Progress in algorithms for solving graph games will therefore allow for the development of more efficient model checkers and contribute to bringing synthesis techniques to practice.

There is a hierarchy among the graph games mentioned earlier, with simple and well known reductions from parity games to mean payoff games, from mean-payoff games to discounted payoff games, and from discounted payoff games to simple stochastic games like the ones from [38], while no reductions are known in the other direction. Therefore, one can solve instances of all these games by using an algorithm for stochastic games. All of these games are in UP and co-UP [17], while no tractable algorithm is known.

Most research has focused on parity games: as the most special class of games, algorithms have the option to use the special structure of their problems, and they are most directly linked to the synthesis and verification problems mentioned earlier. Parity games have thus enjoyed a special status among graph games and the quest for efficient algorithms [13, 11, 26, 38, 7, 37, 27, 4, 3] for solving them has been an active field of research during the last decades, which has received further boost with the arrival of quasi-polynomial techniques [8, 18, 15, 23, 24, 10].

Interestingly, the one class of efficient techniques for solving parity games that does not (yet) have a quasi-polynomial approach is strategy improvement algorithms [25, 29, 35, 6, 30, 14, 33], a class of algorithms closely related to the Simplex for linear programming, known to perform well in practice. Most of these algorithms reduce to mean [6, 30, 33, 5] or discounted [25, 29, 16, 21] payoff games.

A. Achilleos and D. Della Monica (Eds.): Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification (GandALF 2023).
EPTCS 390, 2023, pp. 203–219, doi:10.4204/EPTCS.390.13

With the exception of the case in which the fixed-point of discounted payoff games is explicitly computed [38], all these algorithms share a disappointing feature: they are inherently non-symmetric approaches for solving an inherently symmetric problem. However, some of these approaches have a degree of symmetry. Recursive approaches treat even and odd colours symmetrically, one at a time, but they treat the two players very differently for a given colour. Symmetric strategy improvement [33] runs a strategy improvement algorithms for both players in parallel, using the intermediate results of each of them to inform the updates of the other, but at heart, these are still two intertwined strategy improvement algorithms that, individually, are not symmetric. This is in due to the fact that applying strategy improvement itself symmetrically can lead to cycles [9].

The key contribution of this paper is to devise a new class of algorithms to solve discounted payoff games, which is entirely symmetric. Like strategy improvement algorithms, it seeks to find co-optimal strategies, and improves strategies while they are not optimal. In order to do so, however, it does not distinguish between the strategies of the two players. This seems surprising, as maximising and minimising appear to pull in opposing directions.

Similar to strategy improvement approaches, the new objective improvement approach turns the edges of a game into constraints (here called inequations), and minimises an objective function. However, while strategy improvement algorithms take only the edges in the strategy of one player (and all edges of the other player) into account and then finds the optimal response by solving the resulting one player game, objective improvement always takes all edges into account. The strategies under consideration then form a subset of the inequations, and the goal would be to make them sharp (i.e. as equations), which only works when both strategies are optimal. When they are not, then there is some *offset* for each of the inequations, and the objective is to reduce this offset in every improvement step.

This treats the strategies of both players completely symmetrically.

Organisation of the Paper. The rest of the paper is organised as follows. After the preliminaries (Section 2), we start by outlining our method and use a simple game to explain it (Section 3). We then formally introduce our objective improvement algorithm in Section 4, keeping the question of how to choose a better strategies abstract. Section 5 then discusses how to find better strategies. We finally wrap up with a discussion of our results in Section 6.

2 Preliminaries

A *discounted payoff game* (DPG) is a tuple $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$, where $V = V_{\min} \cup V_{\max}$ are the vertices of the game, partitioned into two disjoint sets $V_{\min}$ and $V_{\max}$, such that the pair (V, E) is a finite directed graph without sinks. The vertices in $V_{\max}$ (*resp*, $V_{\min}$) are controlled by Player Max or maximiser (*resp*, Player Min or minimiser) and $E \subseteq V \times V$ is the edge relation. Every edge has a weight represented by the function $w : E \to \mathbb{R}$, and a *discount factor* represented by the function $\lambda : E \to [0,1)$. When the discount factor is uniform, i.e. the same for every edge, it is represented by a constant value $\lambda \in [0,1)$. For ease of notation, we write w_e and λ_e instead of $w(e)$ and $\lambda(e)$. A *play* on $\mathcal{G}$ from a vertex v is an infinite path, which can be represented as a sequence of edges $\rho = e_0 e_1 e_2 \ldots$ such that, for every $i \in \mathbb{N}^*$, $e_i = (v_i, v_i') \in E$, and, for all $i \in \mathbb{N}, v_{i+1} = v_i'$ and $v_0 = v$. By ρ_i we refer to the i-th edge of the play. The *outcome* of a discounted game $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ for a play ρ is $\mathsf{out}(\rho) = \sum_{i=0}^{\infty} w_{e_i} \prod_{j=0}^{i-1} \lambda_{e_j}$. For games with a constant discount factor, this simplifies in $\mathsf{out}(\rho) = \sum_{i=0}^{\infty} w_{e_i} \lambda^i$.

A positional strategy for Max is a function $\sigma_{\max} : V_{\max} \to V$ that maps each Max vertex to a vertex according to the set of edges, i.e. $(v, \sigma_{\max}(v)) \in E$. Positional Min strategies are defined accordingly, and we call the set of positional Min and Max strategies $\Sigma_{\min}$ and $\Sigma_{\max}$, respectively.

A pair of strategies $\sigma_{\min}$ and $\sigma_{\max}$, one for each player, defines a unique run $\rho(v,\sigma_{\min},\sigma_{\max})$ from each vertex $v \in V$. Discounted payoff games are positionally determined [38]:

$$\sup_{\sigma_{\max}\in\Sigma_{\max}} \inf_{\sigma_{\min}\in\Sigma_{\min}} \mathsf{out}(\rho(v,\sigma_{\min},\sigma_{\max})) = \inf_{\sigma_{\min}\in\Sigma_{\min}} \sup_{\sigma_{\max}\in\Sigma_{\max}} \mathsf{out}(\rho(v,\sigma_{\min},\sigma_{\max}))$$

holds for all $v \in V$, and neither the value, nor the optimal strategy for which it is taken, changes when we allow more powerful classes of strategies that allow for using memory and/or randomisation for one or both players.

The resulting *value of* $\mathscr{G}$, denoted by $\mathsf{val}(\mathscr{G}) : V \to \mathscr{R}$, is defined as

$$\mathsf{val}(\mathscr{G}) : v \mapsto \sup_{\sigma_{\max}\in\Sigma_{\max}} \inf_{\sigma_{\min}\in\Sigma_{\min}} \mathsf{out}(\rho(v,\sigma_{\min},\sigma_{\max})) .$$

The solution to a discounted payoff game is a valuation $\mathsf{val} = \mathsf{val}(\mathscr{G})$ of $\mathscr{G}$ for the vertices such that, for every edge $e = (v,v')$, it holds that[1]

- $\mathsf{val}(v) \le w_e + \lambda_e \mathsf{val}(v')$ if v is a minimiser vertex and
- $\mathsf{val}(v) \ge w_e + \lambda_e \mathsf{val}(v')$ if v is a maximiser vertex.

A positional maximiser (resp. minimiser) strategy σ is optimal if, and only if, $\mathsf{val}(v) = w_{(v,\sigma(v))} + \lambda_{(v,\sigma(v))}\mathsf{val}(\sigma(v))$ holds for all maximiser (resp. minimiser) positions.

Likewise, we define the value of a pair of strategies $\sigma_{\min}$ and $\sigma_{\max}$, denoted $\mathsf{val}(\sigma_{\min},\sigma_{\max}) : V \to \mathscr{R}$, as $\mathsf{val}(\sigma_{\min},\sigma_{\max}) : v \mapsto \mathsf{out}(\rho(v,\sigma_{\min},\sigma_{\max}))$.

As we treat both players symmetrically in this paper, we define a *pair of strategies* $\sigma : V \mapsto V$ whose restriction to $V_{\min}$ and $V_{\max}$ are a minimiser strategy $\sigma_{\min}$ and a maximiser strategy $\sigma_{\max}$, respectively. We then write $\rho(v,\sigma)$ instead of $\rho(v,\sigma_{\min},\sigma_{\max})$ and $\mathsf{val}(\sigma)$ instead of $\mathsf{val}(\sigma_{\min},\sigma_{\max})$.

If both of these strategies are optimal, we call σ a joint *co-optimal* strategy. This is the case if, and only if, $\mathsf{val}(\mathscr{G}) = \mathsf{val}(\sigma)$ holds.

Note that we are interested in the *value* of each vertex, not merely if the value is greater or equal than a given threshold value.

3 Outline and Motivation Example

We start with considering the simple discounted payoff game of Figure 1, assuming that it has some uniform discount factor $\lambda \in [0,1)$. In this game, the minimiser (who owns the right vertex, a, marked by a square), has only one option: she always has to use the self-loop, which earns her an immediate reward of 1. The overall reward the minimiser reaps for a run that starts in her vertex is therefore $1+\lambda+\lambda^2+\ldots = \frac{1}{1-\lambda}$.

The maximiser (who owns the left vertex, b, marked by a circle) can choose to either use the self-loop, or to move to the minimiser vertex (marked by a square), both yielding no immediate reward.

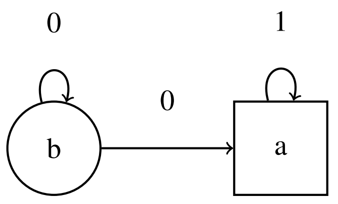


Figure 1: A discounted Payoff Game. Maximiser vertices are marked by a circle, minimizer ones by a square.

[1]These are the constraints represented in H in Section 4.

If the maximiser decides to stay forever in his vertex (using the self-loop), his overall reward in the play that starts at (and, due to his choice, always stays in) his vertex, is 0. If he decides to move on to the minimiser vertex the n^{th}-time, then the reward is $\frac{\lambda^n}{1-\lambda}$.

The optimal decision of the maximiser is therefore to move on the first time, which yields him the maximal reward of $\frac{\lambda}{1-\lambda}$. Every vertex v has some outgoing edge(s) $e=(v,v')$ where $\mathsf{val}(v)=w_e+\lambda_e\mathsf{val}(v')$ holds [38]; these edges correspond to the optimal decisions for the respective player.

For our running example game of Figure 1 with a fixed discount factor $\lambda\in[0,1)$, the inequations are

1. $\mathsf{val}(a)\leq 1+\lambda\mathsf{val}(a)$ for the self-loop of the minimiser vertex;
2. $\mathsf{val}(b)\geq\lambda\mathsf{val}(b)$ for the self-loop of the maximiser vertex; and
3. $\mathsf{val}(b)\geq\lambda\mathsf{val}(a)$ for the transition from the maximiser to the minimiser vertex.

The unique valuation that satisfies these inequations and produces a sharp inequation (i.e. satisfied as equation) for some outgoing edge of each vertex assigns $\mathsf{val}(a)=\frac{1}{1-\lambda}$ and $\mathsf{val}(b)=\frac{\lambda}{1-\lambda}$. This valuation also defines the optimal strategies of the players (to stay for the minimiser, and to move on for the maximiser).

Solving a discounted payoff game means finding this valuation and/or these strategies.

We discuss a symmetric approach to find this unique valuation. Our approach adjusts linear programming in a natural way that treats both players symmetrically: we maintain the set of inequations for the complete time, while approximating the goal of "one equality per vertex" by the objective function. To do that, we initially fix an *arbitrary* outgoing edge for every vertex (a strategy), and minimise the sum of the distances between the left and right side of the inequations defined by these edges, which we call the *offset* of this edge. This means, for an edge $e=(v,v')$, to minimise the difference of $\mathsf{val}(v)$ (left side of the inequation) and $w_e+\lambda_e\mathsf{val}(v')$ (right side).

To make this clear, we consider again the example of Figure 1 and use both self loops as the strategies for the players fixed at the beginning in our running example. The offset for the selected outgoing edge of the minimiser vertex a is equal to $1-(1-\lambda)\mathsf{val}(a)$, while the offset for the selected outgoing edge of the maximiser vertex b is equal to $(1-\lambda)\mathsf{val}(b)$. The resulting overall objective consists, therefore, in minimising the value $1-(1-\lambda)\mathsf{val}(a)+(1-\lambda)\mathsf{val}(b)$.

This term is always non-negative, since it corresponds to the sum of the edges contributions that are all non-negative. Moreover, when only optimal strategies are selected to form this objective function, the value 0 can be taken, and where it is taken, it defines the correct valuation.

As the maximiser's choice to take the self-loop is not optimal, the resulting objective function the strategies define, that is $1-(1-\lambda)\mathsf{val}(a)+(1-\lambda)\mathsf{val}(b)$, cannot reach 0. But let us take a look at what an optimal solution w.r.t. this objective function looks like.

Optimal solutions can be taken from the corners of the polytope defined by the inequations. In this case, the optimal solution (w.r.t. this initial objective function) is defined by making inequations (1) and (3) sharp: this provides the values $\mathsf{val}(a)=\frac{1}{1-\lambda}$ and $\mathsf{val}(b)=\frac{\lambda}{1-\lambda}$; the objective function takes the value λ at this point.

For comparison, in the other corner of the polytope, defined by making inequations (2) and (3) sharp, we obtain the values $\mathsf{val}(a)=\mathsf{val}(b)=0$; the objective function takes the value 1 at this point. Finally, if we consider the last combination, making (1) and (2) sharp provides the values $\mathsf{val}(a)=\frac{1}{1-\lambda}$ and $\mathsf{val}(b)=0$, so that inequation (3) is not satisfied; this is therefore not a corner of the polytope.

Thus, in this toy example, while selecting the wrong edge cannot result in the objective function taking the value 0, we still found the optimal solution. In general, we might need to update the objective function. To update the objective function, we change the outgoing edges of some (or all) vertices, such that the overall value of the objective function goes down. Note that this can be done not only when the linear program returns an optimal solution, but also during its computation. For example, when using a simplex method, updating the objective function can be used as an alternative pivoting rule at any point of the traversal of the polytope.

Unfortunately, the case in which the valuation returned as solution is computed using an objective function based on non-optimal strategies, is not the general case. The simplest way of seeing this is to use different discount factors for the game of Figure 1[2], let's say $\frac{1}{3}$ for the self-loop of the maximiser vertex and $\frac{2}{3}$ for the other two transitions, so that the three equations are: (1) $\mathsf{val}(a) \leq 1 + \frac{2}{3}\mathsf{val}(a)$, (2) $\mathsf{val}(b) \geq \frac{2}{3}\mathsf{val}(b)$, and (3) $\mathsf{val}(b) \geq \frac{1}{3}\mathsf{val}(a)$. Making the adjusted inequations (2) and (3) sharp still results in the values $\mathsf{val}(a) = \mathsf{val}(b) = 0$, and the objective function still takes the value of 1. While making inequations (1) and (3) sharp provides the values $\mathsf{val}(a) = 3$ and $\mathsf{val}(b) = 2$; the objective function takes the value $\frac{4}{3}$ at this point. Finally, if we consider the last combination, making (1) and (2) sharp still conflicts with inequation (3).

Thus, $\mathsf{val}(a) = \mathsf{val}(b) = 0$ would be the optimal solution for the given objective function, which is not the valuation of the game. We will then update the candidate strategies so that the sum of the offsets goes down.

3.1 Comparison to strategy improvement

The closest relative to our new approach are strategy improvement algorithms. Classic strategy improvement approaches solve this problem of finding the valuation of a game (and usually also co-optimal strategies) by (1) fixing a strategy for one of the players (we assume w.l.o.g. that this is the maximiser), (2) finding a valuation function for the one player game that results from fixing this strategy (often together with an optimal counter strategy for their opponent), and (3) updating the strategy of the maximiser by applying local improvements. This is repeated until no local improvements are available, which entails that the constraint system is satisfied.

For Step (2) of this approach, we can use linear programming, which does invite a comparison to our technique. The linear program for solving Step (2) would not use all inequations: it would, instead, replace the inequations defined by the currently selected edges of the maximiser by equations, while dropping the inequations defined by the other maximiser transitions. The objective function would then be to minimise the values of all vertices while still complying with the remaining (in)equations.

Thus, while in our novel symmetric approach the constraints remain while the objective is updated, in strategy improvement the objective remains, while the constraints are updated.

Moreover, the players and their strategies are treated quite differently in strategy improvement algorithms: while the candidate strategy of the maximiser results in making the inequations of the selected edges sharp (and dropping all other inequations of maximiser edges), the optimal counter strategy is found by minimising the objective. This is again in contrast to our novel symmetric approach, which treats both players equally.

A small further difference is in the valuations that can be taken: the valuations that strategy improvement algorithms can take are the valuations of strategies, while the valuations our objective improvement algorithm can take on the way are the corners of the polytope defined by the inequations. Except for the only intersection point between the two (the valuation of the game), these corners of the polytope do not relate to the value of strategies. Table 1 summarises these observations.

[2]Note that we can also replace the transitions with a smaller discount factor by multiple transitions with a larger discount factor. This would allow for keeping the discount factor uniform, but needlessly complicate the discussion and inflate the size of the example.

	Objective Improvement	**Strategy Improvement**
players	symmetric treatment	asymmetric treatment
constraints	remain the same: one inequation per edge	change: one inequation for each edge defined by the current strategy for the strategy player, one inequation for every edge of their opponent
objective	minimise errors for selected edges	maximise values
update	objective: one edge for each vertex	strategy: one edge for each vertex of the strategy player
valuations	corners of polytope	defined by strategies

Table 1: A comparison of the novel objective improvement with classic strategy improvement.

4 General Objective Improvement

In this section, we present the approach outlined in the previous section more formally, while keeping the most complex step – updating the candidate strategy to one which is *better* in that it defines an optimisation function that can take a smaller value – abstract. (We will turn back to the question of how to find better strategies in Section 5.) This allows for discussing the principal properties more clearly.

A general outline of our *objective improvement* approach is based on this algorithm:

Algorithm 1: Objective Improvement

```
input : A discounted payoff game $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$
output: The valuation val of $\mathcal{G}$
1 $H \leftarrow$ Inequations($\mathcal{G}$)
2 $\sigma \leftarrow$ ChooseInitialStrategies($\mathcal{G}$)
3 while true do
4     $f_\sigma \leftarrow$ ObjectiveFunction($\mathcal{G}, \sigma$)
5     val $\leftarrow$ LinearProgramming($H, f_\sigma$)
6     if $f_\sigma(\text{val}) = 0$ then
          return val
      end
7     $\sigma \leftarrow$ ChooseBetterStrategies($\mathcal{G}, \sigma$)
  end
```

Before describing the procedures called by the algorithm, we first outline the principle.

When running on a discounted payoff game $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$, the algorithm uses a set of inequations defined by the edges of the game and the owner of the source of each edge. This set of inequations denoted by H contains one inequation for each edge and (different to strategy improvement approaches whose set of inequations is a subset of H) H never changes.

The inequations from H are computed by a function called Inequations that, given the discounted game $\mathcal{G}$, returns the set made of one inequation per edge $e = (v, v') \in E$, defined as follows:

$$I_e = \begin{cases} \text{val}(v) \geq w_e + \lambda_e \text{val}(v') & \text{if } v \in V_{\max} \\ \text{val}(v) \leq w_e + \lambda_e \text{val}(v') & \text{otherwise .} \end{cases}$$

The set $H = \{I_e \mid e \in E\}$ is defined as the set of all inequations for the edges of the game.

The algorithm also handles strategies for both players, treated as a single strategy σ. They are initialised (for example randomly) by the function ChooseInitialStrategies.

This joint strategy is used to define an objective function f_σ by calling function ObjectiveFunction, whose value on an evaluation val is: $f_\sigma(\text{val}) = \sum_{v \in V} f_\sigma(\text{val}, v)$ with the following objective function components:

$$f_\sigma(\mathsf{val}, v) = \mathsf{offset}(\mathsf{val}, (v, \sigma(v)))\ ,$$

where the offset of an edge (v, v') for a valuation is defined as follows:

$$\mathsf{offset}(\mathsf{val}, (v, v')) = \begin{cases} \mathsf{val}(v) - (w_{(v,v')} + \lambda_{(v,v')}\mathsf{val}(v')) & \text{if } v \in V_{\max} \\ (w_{(v,v')} + \lambda_{(v,v')}\mathsf{val}(v')) - \mathsf{val}(v) & \text{otherwise} \end{cases}$$

This objective function f_σ is given to a linear programming algorithm, alongside with the inequations set H. We underline that, due to the inequation to $I_{(v,v')}$, the value of $\mathsf{offset}(\mathsf{val}, (v, v'))$ is non-negative for all $(v, v') \in E$ in any valuation val (optimal or not) that satisfies the system of inequations H. We put a further restriction on val in that we require it to be the solution to a *basis* $\mathbf{b}$ in H. Such a basis consists of $|V|$ inequations that are satisfied sharply (i.e. as equations), such that these $|V|$ equations uniquely define the values of all vertices. We refer to this valuation as the evaluation of $\mathbf{b}$, denoted $\mathsf{val}(\mathbf{b})$.

The call $\mathsf{LinearProgramming}(H, f_\sigma)$ to some linear programming algorithm then returns a valuation val of the vertices that minimise f_σ while satisfying H, and for convenience require this valuation to also be $\mathsf{val}(\mathbf{b})$ for some base $\mathbf{b}$ of H. (Note that the simplex algorithm, for example, only uses valuations of this form in every step.) We call this valuation a *valuation associated to σ*.

Observation 1. *At Line 6 of Algorithm 1, the value of $f_\sigma(\mathsf{val})$ is non-negative.*

We say that a valuation val *defines* strategies of both players if, for every vertex $v \in V$, the inequation of (at least) one of the outgoing edges of v is sharp. These are the strategies defined by using, for every vertex $v \in V$, an outgoing edge for which the inequation is sharp. Note that there can be more than one of these inequations for some of the vertices.

Observation 2. *If, for a solution val of H, $f_\sigma(\mathsf{val}) = 0$ holds, then, for every vertex $v \in V$, the inequation $I_{(v,\sigma(v))}$ for the edge $(v, \sigma(v))$ is sharp, and val therefore defines strategies for both players, those defined by σ, for example.*

We can use, alternatively, $f_\sigma(\mathsf{val}) = 0$ as a termination condition, as shown in Algorithm 1, since in this case σ must define co-optimal strategies.

Theorem 1. *If σ describes co-optimal strategies, then $f_\sigma(\mathsf{val}) = 0$ holds at Line 6 of Algorithm 1. If val defines strategies for both players joint in σ at Line 6 of Algorithm 1, then σ is co-optimal and val is the valuation of $\mathcal{G}$.*

Proof. The valuation $\mathsf{val} = \mathsf{val}(\mathcal{G})$ of the game is the unique solution of H for which, for all vertices v, the inequation to (at least) one of the outgoing edges of v is sharp, and the edges for which they are sharp describe co-optimal strategies. The valuation of the game is thus the only valuation that *defines* strategies for both players, which shows the second claim.

Moreover, if σ describes co-optimal strategies, then $f_\sigma(\mathsf{val}) = 0$ holds for $\mathsf{val} = \mathsf{val}(\mathcal{G})$ (and for this valuation only), which establishes the first claim. □

The theorem above ensures that, in case the condition at Line 6 holds, the algorithm terminates and provides the value of the game that then allows us to infer optimal strategies of both players. Otherwise we have to improve the objective function and make another iteration of the while loop. At Line 7, $\mathsf{ChooseBetterStrategies}$ can be any procedure that, for $f_\sigma(\mathsf{val}) \neq 0$, provides a pair of strategy σ' *better* than σ as defined in the following subsection.

Better strategies A strategy σ' for both players is *better* than a strategy σ if, and only if, the minimal value of the objective function $f_{\sigma'}$ (computed by $\mathsf{LinearProgramming}(H, f_{\sigma'})$) is strictly lower than the minimal value of the objective function for f_σ (computed by $\mathsf{LinearProgramming}(H, f_\sigma)$). Formally, $f_{\sigma'}(\mathsf{val}') < f_\sigma(\mathsf{val})$.

While we discuss how to implement this key function in the next section, we observe here that the algorithm terminates with a correct result with any implementation that chooses a better objective function in each round: correctness is due to it only terminating when val *defines* strategies for both players, which implies (cf. Theorem 1) that val is the valuation of $\mathscr{G}$ ($\mathsf{val} = \mathsf{val}(\mathscr{G})$) and all strategies defined by val are co-optimal. Termination is obtained by a finite number of positional strategies: by Observation 1, the value of the objective function of all of them is non-negative, while the objective function of an optimal solution to co-optimal strategies is 0 (cf. Theorem 1), which meets the termination condition of Line 6 (cf. Observation 2).

Corollary 1. *Algorithm 1 always terminates with the correct value.*

5 Choosing Better Strategies

In this section, we will discuss sufficient criteria for efficiently finding a procedure that implements $\mathsf{ChooseBetterStrategies}$. For this, we make four observations described in the next subsections:

1. All local improvements can be applied. A strategy σ' is a local improvement to a strategy σ if $f_{\sigma'}(\mathsf{val}) < f_\sigma(\mathsf{val})$ holds for the current valuation val (Section 5.1).
2. If the current valuation val does not *define* a pair of strategies σ for both players and has no local improvements, then a better strategy σ' can be found by applying only switches from and to edges that already have offset 0 (Section 5.2).
3. The improvement mentioned in the previous point can be found for special games (the sharp and improving games defined in Section 5.3) by trying a single edge switch.
4. Games can almost surely be made sharp and improving by adding random noise that retains optimal strategies (Section 5.4).

Together, these four points provide efficient means for finding increasingly better strategies, and thus to find the co-optimal strategies and the valuation of the discounted payoff game.

As a small side observation, when using a simplex based technique to implement $\mathsf{LinearProgramming}$ at Line 5 of Algorithm 1, then the pivoting of the objective function from point (1.) and the pivoting of the base can be mixed (this will be discussed in Section 5.5).

5.1 Local Improvements

The simplest (and most common) case of creating better strategies σ' from a valuation for the objective f_σ for a strategy σ is to consider *local improvements*. Much like local improvements in strategy iteration approaches, local improvements consider, for each vertex v, a successor $v' \neq \sigma(v)$, such that $\mathsf{offset}(\mathsf{val}, (v, v')) < \mathsf{offset}(\mathsf{val}, (v, \sigma(v)))$ for the current valuation val, which is optimal for the objective function f_σ.

To be more general, our approach does not necessarily requires to select only local improvements, but it can work with global improvements, though we cannot see any practical use of choosing differently. For instance, if we treat the function as a global improvement approach, we can update the value for a vertex v such that it increases by 1 and update the value of another vertex v' such that it decreases by 2. The overall value of the function will decrease, even if locally some components increased their value. Interestingly, this cannot be done with a strategy improvement approach, as it requires to always locally improve the value of each vertex when updating.

Lemma 1. *If* val *is an optimal valuation for the linear programming problem at Line 5 of Algorithm 1 and* $f_{\sigma'}(\mathsf{val}) < f_\sigma(\mathsf{val})$*, then* σ' *is better than* σ*.*

Proof. The valuation val is, being an optimal solution for the objective f_σ, a solution to the system of inequations H. For a solution val' which is optimal for $f_{\sigma'}$, we thus have $f_{\sigma'}(\mathsf{val}') \leq f_{\sigma'}(\mathsf{val}) < f_\sigma(\mathsf{val})$, which implies that σ' is better than σ accordingly to notion of *better* strategy provided at the end of Section 4. □

5.2 No Local Improvements

The absence of local improvements means that, for all vertices $v \in V$ and all outgoing edges $(v,v') \in E$, $\mathsf{offset}(\mathsf{val},(v,v')) \geq \mathsf{offset}(\mathsf{val},(v,\sigma(v)))$.

We define for a valuation val optimal for a f_σ (like the val produced in line 5 of Algorithm 1):

- $S^\sigma_{\mathsf{val}} = \{(v,v') \in E \mid \mathsf{offset}(\mathsf{val},(v,v')) = \mathsf{offset}(\mathsf{val},(v,\sigma(v)))\}$ as the set of *stale* edges; naturally, every vertex has at least one outgoing stale edge: the one defined by σ;
- $E_{\mathsf{val}} = \{(v,v') \in E \mid \mathsf{offset}(\mathsf{val},(v,v')) = 0\}$ as the set of edges, for which the inequation for val is sharp; in particular, all edges in the base of H that defines val are sharp (and stale); and
- E^σ_{val} as any set of edges between E_{val} and S^σ_{val} (i.e. $E_{\mathsf{val}} \subseteq E^\sigma_{\mathsf{val}} \subseteq S^\sigma_{\mathsf{val}}$) such that E^σ_{val} contains an outgoing edge for every vertex $v \in V$; we are interested to deal with sets that retain the game property that every vertex has a successor, we can do that by adding (non sharp) stale edges to E_{val}.

Note that S^σ_{val} is such a set, hence, an adequate set is easy to identify. We might, however, be interested in keeping the set small, and choosing the edges defined by E_{val} plus one outgoing edge for every vertex v that does not have an outgoing edge in E_{val}. The most natural solutions is to choose the edge $(v,\sigma(v)) \in E^\sigma_{\mathsf{val}}$ defined by σ for each such vertex v.

Observation 3. *If* $\mathscr{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ *is a DPG and* σ *a strategy for both players such that* val *is an optimal solution for the objective* f_σ *to the system of inequations* H*, then* $\mathscr{G}' = (V_{\min}, V_{\max}, E^\sigma_{\mathsf{val}}, w, \lambda)$ *is also a DPG.*

This simply holds because every vertex $v \in V$ retains at least one outgoing transition.

Lemma 2. *Let* $\mathscr{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ *be a DPG,* σ *a strategy for both players,* val *an optimal solution returned at Line 5 of Algorithm 1 for* f_σ*, and let there be no local improvements of* σ *for* val*. If* val *does not define strategies of both players, then there is a better strategy* σ' *such that, for all* $v \in V$*,* $(v,\sigma'(v)) \in E^\sigma_{\mathsf{val}}$*.*

Proof. By Observation 3, $\mathscr{G}' = (V_{\min}, V_{\max}, E^\sigma_{\mathsf{val}}, w, \lambda)$ is a DPG. Let val' be the valuation of $\mathscr{G}'$, and σ' be the strategies for the two players defined by it.

If val' is also a valuation of $\mathscr{G}$, then we are done. However, this need not be the case, as the system of inequations H' for $\mathscr{G}'$ is smaller than the set of inequations H for $\mathscr{G}$, so val' might violate some of the inequations that are in H, but not in H'. Given that val' is a valuation for $\mathscr{G}'$, it satisfies all inequations in

H'. Moreover, since val also satisfies all inequations of H', it follows that the same inequations hold for every convex combination of val and val'.

We now note that the inequations of H that are not in H' are not sharp for val. Thus, there is an $\varepsilon \in (0,1]$ such that the convex combination $\mathsf{val}_\varepsilon = \varepsilon \cdot \mathsf{val}' + (1-\varepsilon)\mathsf{val}$ is a solution to those inequations.

We now have $f_{\sigma'}(\mathsf{val}) = f_\sigma(\mathsf{val}) > 0$ and $f_{\sigma'}(\mathsf{val}') = 0$. For an optimal solution val'' of H for the objective $f_{\sigma'}$, this provides $f_{\sigma'}(\mathsf{val}'') \leq f_{\sigma'}(\mathsf{val}_\varepsilon) < f_\sigma(\mathsf{val})$.

Therefore σ' is better than σ. □

When using the most natural choice, $E^\sigma_{\mathsf{val}} = E_{\mathsf{val}} \cup \{(v,\sigma(v)) \mid v \in V\}$, this translates in keeping all transitions, for which the offset is *not* 0, while changing some of those, for which the offset already is 0. This is a slightly surprising choice, since to progress one has to improve on the transitions whose offset is positive, and ignore those with offset 0.

5.3 Games with Efficient Objective Improvement

In this subsection, we consider sufficient conditions for finding better strategies efficiently. Note that we only have to consider cases where the termination condition (Line 6 of Algorithm 1) is not met.

The simplest condition for efficiently finding better strategies is the existence of local improvements. (In particular, it is easy to find, for a given valuation val, strategies σ' for both players such that $f_{\sigma'}(\mathsf{val}) \leq f_{\sigma''}(\mathsf{val})$ holds for all strategies σ''). When there are local improvements, we can obtain a better strategy simply by applying them.

This leaves the case in which there are no local improvements, but where val also does not *define* strategies for the two players. We have seen that we can obtain a better strategy by only swapping edges, for which the inequations are sharp (Lemma 2).

We will now describe two conditions that, when both met, will allow us to efficiently find better strategies: that games are *sharp* and *improving*.

Sharp games. To do this efficiently, it helps if there are always $|V|$ inequations that are sharp: there must be at least $|V|$ of them for a solution returned by the simplex method, as it is the solution defined by making $|V|$ inequations sharp (they are called the base), and requiring that there are exactly $|V|$ many of them means that the valuation we obtain defines a base. We call such a set of inequations H, and games that define them *sharp DPGs*.

Improving games. The second condition, which will allow us to identify better strategies efficiently, is to assume that, for every strategy σ for both players, if a valuation val defined by a base is not optimal for f_σ under the constraints H, then there is a single base change that improves it. We call such sharp DPGs *improving*.

We call a valuation val' whose base can be obtained from that of val by a single change to the base of val a neighbouring valuation to val. We will show that, for improving games, we can sharpen the result of Lemma 2 so that the better strategy σ' also guarantees $f_{\sigma'}(\mathsf{val}') < f_\sigma(\mathsf{val})$ for some neighbouring valuation val' to val.

This allows us to consider $O(|E|)$ base changes and, where they define a valuation, to seek optimal strategies for a given valuation. Finding an optimal strategy for a given valuation is straightforward.

Theorem 2. *Let $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ be an improving DPG, σ a strategy for both players,* val *an optimal solution returned at Line 5 of Algorithm 1 for f_σ, and let there be no local improvements of σ for* val*. Then there is (at least) one neighbouring valuation* val'' *to* val *such that there is a better strategy σ' that satisfies $f_{\sigma'}(\mathsf{val}'') < f_\sigma(\mathsf{val})$.*

Such a strategy σ' is better than σ, and it can be selected in a way that $(v,\sigma'(v)) \in E^{\sigma}_{\mathsf{val}}$ holds for all $v \in V$ for a given set $E^{\sigma}_{\mathsf{val}}$.

Proof. We apply Lemma 2 for $E^{\sigma}_{\mathsf{val}} = E_{\mathsf{val}} \cup \{(v,\sigma(v) \mid v \in V\}$ and use the resulting better strategy σ' for this set $E^{\sigma}_{\mathsf{val}}$. Let val' be the optimal solution for $f_{\sigma'}$ that satisfies the constraints H defined by $\mathcal{G}$. Note that since σ' is better than σ by Lemma 2, we have that $f_{\sigma'}(\mathsf{val}') < f_{\sigma}(\mathsf{val})$ and $f_{\sigma}(\mathsf{val}) \leq f_{\sigma}(\mathsf{val}')$.

We now consider an arbitrary sequence of evaluations $\mathsf{val} = \mathsf{val}_0, \mathsf{val}_1, \ldots, \mathsf{val}_n = \mathsf{val}'$ along the edges of the simplex from val to val', such that the value of the new objective function $f_{\sigma'}$ only decreases. Note that such a path must exist, as the simplex algorithm would pick it.

The sharpness of $\mathcal{G}$ implies that $\mathsf{val}_1 \neq \mathsf{val}_0$, and considering that $\mathcal{G}$ is improving provides $f_{\sigma'}(\mathsf{val}_1) < f_{\sigma'}(\mathsf{val}_0)$.

Thus, when only applying a single base change, we move to a fresh value, val_1, such that $f_{\sigma'}(\mathsf{val}_1) < f_{\sigma}(\mathsf{val})$ for some σ'.

Note that σ' was supplied by Lemma 2, so that $(v,\sigma'(v)) \in E^{\sigma}_{\mathsf{val}}$ holds. □

While we can restrict the selection of σ' to those that comply with the restriction $(v,\sigma'(v)) \in E^{\sigma}_{\mathsf{val}}$, there is no particular reason for doing so; as soon as we have a neighbouring valuation val', we can identify a pair of strategies σ' for which $f_{\sigma'}(\mathsf{val}')$ is minimal, and select σ' if $f_{\sigma'}(\mathsf{val}') < f_{\sigma}(\mathsf{val})$ holds.

5.4 Making Games Sharp and Improving

Naturally, not every game is improving, or even sharp. In this subsection, we first discuss how to almost surely make games sharp by adding sufficiently small random noise to the edge weights, and then discuss how to treat games that are not improving by assigning randomly chosen factors, with which the offsets of edges are weighted. Note that these are 'global' adjustments of the game that only need to be applied once, as it is the game that becomes sharp and improving, respectively.

Starting with the small noise to add on the edge weights, we first create notation for expressing how much we can change edge weights, such that joint co-optimal strategies of the resulting game are joint co-optimal strategies in the original game. To this end, we define the *gap* of a game.

Gap of a game. For a DPG $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$, we call $\lambda^* = \max\{\lambda_e \mid e \in E\}$ its contraction. For a joint strategy σ that is *not* co-optimal, we call $\gamma_\sigma = -\min\{\mathsf{offset}(\mathsf{val}(\sigma), e) \mid e \in E\}$; note that $\gamma_\sigma > 0$ holds. We call the minimal[3] such value γ the *gap of* $\mathcal{G}$.

Note that $\mathsf{val}(\sigma)$ is the valuation of the joint strategy σ, not the outcome of the optimisation problem. This is because we use the gap of the game to argue that a non-co-optimal strategy remains non-co-optimal after a small distortion of the edge weights, so that the value of the joint strategy itself is useful. (It is much easier to access than the result of optimisation.) This also implies that the offsets can be negative.

We now use the gap of a game γ to define the magnitude of a change to all weights, such that all strategies that used to have a gap still have one.

Lemma 3. *Let $\mathcal{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ be a DPG with contraction λ^* and gap γ, and let $\mathcal{G}' = (V_{\min}, V_{\max}, E, w', \lambda)$ differ from $\mathcal{G}$ only in the edge weights such that, for all $e \in E$, $|w_e - w'_e| \leq \frac{1-\lambda^*}{3}\gamma$ holds. Then a joint co-optimal strategy from $\mathcal{G}'$ is also co-optimal for $\mathcal{G}$.*

[3]This skips over the case where all strategies are co-optimal, but that case is trivial to check and such games are trivial to solve, so that we ignore this case in this subsection.

Proof. The small weight disturbance, $|w_e - w'_e| \leq \frac{1-\lambda^*}{3}\gamma$ for all $e \in E$, immediately provides a small difference in the valuation: for all joint strategies σ, we have for $\mathsf{val} = \mathsf{val}(\sigma)$ on $\mathscr{G}$, and $\mathsf{val}' = \mathsf{val}(\sigma)$ on $\mathscr{G}'$, that $|\mathsf{val}(v) - \mathsf{val}'(v)| \leq \frac{1}{1-\lambda^*}\frac{1-\lambda^*}{3}\gamma = \frac{\gamma}{3}$, using the definition of $\mathsf{val}(\sigma)$ and triangulation.

More precisely, as σ defines a run $\rho = e_0e_1e_2\ldots$, and we have $\mathsf{val}(v) = \mathsf{out}(\rho) = \sum_{i=0}^{\infty} w_{e_i}\prod_{j=0}^{i-1}\lambda_{e_j}$ and $\mathsf{val}'(v) = \sum_{i=0}^{\infty} w'_{e_i}\prod_{j=0}^{i-1}\lambda_{e_j}$. This provides

$$|\mathsf{val}(v) - \mathsf{val}'(v)| \leq \sum_{i=0}^{\infty}|w_{e_i} - w'_{e_i}|\prod_{j=0}^{i-1}\lambda_{e_j} < \frac{1-\lambda^*}{3}\gamma\sum_{i=0}^{\infty}\prod_{j=0}^{i-1}\lambda_{e_j} \leq \frac{1-\lambda^*}{3}\gamma\sum_{i=0}^{\infty}(\lambda^*)^i = \frac{\gamma}{3}.$$

If σ is not co-optimal for $\mathscr{G}$, we have an edge e with $-\mathsf{offset}(\mathsf{val}, e) = \gamma_\sigma \geq \gamma$. Triangulation provides

$$|\mathsf{offset}(\mathsf{val}', e) - \mathsf{offset}(\mathsf{val}, e)| < \frac{1+\lambda^*}{3}\gamma$$

and (using offset' to indicate the use of w'_e for $\mathscr{G}'$ instead of w_e for $\mathscr{G}$),

$$|\mathsf{offset}'(\mathsf{val}', e) - \mathsf{offset}(\mathsf{val}, e)| \leq \frac{2+\lambda^*}{3}\gamma < \gamma \leq \gamma_\sigma\ ,$$

which, together with the fact that $-\mathsf{offset}(\mathsf{val}, e) \geq \gamma$, provides $\mathsf{offset}'(\mathsf{val}', e) < 0$.

Thus, σ is not co-optimal for $\mathscr{G}'$. □

Lemma 4. *Given a DPGs $\mathscr{G} = (V_{\min}, V_{\max}, E, w, \lambda)$, the DPG $\mathscr{G}' = (V_{\min}, V_{\max}, E, w', \lambda)$ resulting from $\mathscr{G}$ by adding independently uniformly at random drawn values from an interval $(-\varepsilon, \varepsilon)$ to every edge weight, will almost surely result in a sharp game.*

Proof. There are only finitely many bases, and it suffices to compare two arbitrary but fixed ones, say b_1 and b_2.

As they are different, there will be one edge $e = (v, v')$ that occurs in b_1, but not in b_2. As all weight disturbances are drawn independently, we assume without loss of generality that the weight disturbance to this edge is drawn last.

Now, the valuation val_2 defined by b_1 does not depend on this final draw. For val_2, there is a value $w'_e = \mathsf{val}_2(v) - \lambda_e\mathsf{val}_2(v')$ that defines the weight w'_e e would need to have such that the inequation for e is sharp.

For the valuation val_1 defined by b_1 to be equal to val_2, the weight for the edge e (after adding the drawn distortion) needs to be exactly w'_e. There is at most one value for the disturbance that would provide for this, and this disturbance for weight of e is sampled with a likelihood of 0. □

Putting these two results together, we get:

Corollary 2. *Given a pair of DPGs $\mathscr{G} = (V_{\min}, V_{\max}, E, w, \lambda)$ with contraction λ^* and gap γ, and $\mathscr{G}' = (V_{\min}, V_{\max}, E, w', \lambda)$ obtained from $\mathscr{G}$ by adding independently uniformly at random drawn values from an interval $(-\varepsilon, \varepsilon)$ to every edge weight, for some $\varepsilon \leq \frac{1-\lambda^*}{3}\gamma$, then, a joint co-optimal strategy from $\mathscr{G}'$ is also co-optimal for $\mathscr{G}$, and $\mathscr{G}'$ is almost surely sharp.*

Note that we can estimate the gap cheaply when all coefficients in $\mathscr{G}$ are rational. The gap is defined as the minimum of γ_σ over non-co-optimal joint strategies σ, and we start by fixing such a joint strategy.

For a given $v \in V$, σ defines a run $\rho = e_0e_1e_2\ldots$ in the form of a "lasso path", which consists of a (possibly empty) initial path $e_0, \ldots, e_k$, followed by an infinitely often repeated cycle $e'_0, \ldots, e'_\ell$, where the only vertex that occurs twice on the path $e_0,\dots,e_k,e'_0,\dots,e'_\ell$ is the vertex reached before e'_0 and after e'_ℓ, while all other vertices occur only once.

In this run, an edge e_i occurs only once and contributes $\prod_{j=0}^{i-1}\lambda_{e_j}w_{e_i}$ to the value of this run, while an edge e'_i occurs infinitely often and contributes the value $\frac{\prod_{j=0}^{k}\lambda_{e_j}\cdot\prod_{j=0}^{i-1}\lambda_{e'_j}}{1-\prod_{j=0}^{\ell}\lambda_{e'_j}}w_{e'_i}$ to the value of the run.

Now all we need to do is to find a common denominator. To do this, let $\mathsf{denom}(r)$ be the denominator of a rational number r. It is easy to see that

$$\mathsf{common} = \prod_{v\in V}\mathsf{denom}(\lambda_{(v,\sigma(v))})^2\cdot\mathsf{denom}(w_{(v,\sigma(v))})$$

is a common denominator of the contributions of all of these weights, and thus a denominator that can be used for the sum.

Looking at an edge e that defines γ_σ, then γ_σ can be written with the denominator

$$\mathsf{common}\cdot\mathsf{denom}(\lambda_e w_e)\ .$$

Obviously, the nominator is at least 1.

Estimating γ can thus be done by using the highest possible denominators available in $\mathcal{G}$. The representation of the resulting estimate γ is polynomial in the size of $\mathcal{G}$.

Biased sum of offsets. We modify sharp games that are not improving. This can be done by redefining the function offset as follows:

$$\mathsf{offset}'(\mathsf{val},(v,v')) = \alpha_{(v,v')}\cdot\mathsf{offset}(\mathsf{val},(v,v'))$$

Such offset are defined for every edge $e=(v,v')$, where all α_e are positive numbers, which we call the *offset factor*. Based on this change, we re-define all offset definitions and the objective function with a primed version that uses these positive factors.

Theorem 3. *Let $\mathcal{G}=(V_{\min},V_{\max},E,w,\lambda)$ be an improving DPG for a given set of positive numbers $\{\alpha_e>0\mid e\in E\}$, σ a strategy for both players,* val *an optimal solution returned at Line 5 of Algorithm 1 for the adjusted function f'_σ, and let there be no local improvements of σ for* val. *Then there is neighbouring valuation* val'' *to* val *such that there is a better strategy σ' that satisfies $f_{\sigma'}(\mathsf{val}'')<f_\sigma(\mathsf{val})$.*

Such a strategy σ' is better than σ, and it can be selected in a way that $(v,\sigma'(v))\in E^\sigma_{\mathsf{val}}$ holds for all $v\in V$ for a given set E^σ_{val}.

Proof. All proofs can be repeated verbatim with the new offset definition. □

Theorem 4. *If each offset factor α_e is selected independently uniformly at random from a bounded interval of positive numbers*[4], *then a sharp DPG $\mathcal{G}=(V_{\min},V_{\max},E,w,\lambda)$ is almost surely improving for a sampled set of positive numbers $\{\alpha_e>0\mid e\in E\}$.*

Proof. We show the claim for two arbitrary but fixed valuations val_1 and val_2 defined by two different bases, b_1 and b_2, respectively, that satisfy the inequations in H, and an arbitrary but fixed adjusted f_σ. As there are finitely many bases and finitely many joint strategies, satisfying the requirement almost surely for them entails that the requirement is satisfied almost surely for the game.

As $\mathcal{G}$ is sharp, we have $\mathsf{val}_1\neq\mathsf{val}_2$.

[4] or from any other distribution over positive numbers that has 0 weight for all individual points

We first assume for contradiction that $\mathsf{offset}(\mathsf{val}_1,(v,\sigma(v))) = \mathsf{offset}(\mathsf{val}_2,(v,\sigma(v)))$ holds for all $v \in V$. We pick a vertex v such that $|\mathsf{val}_1(v)-\mathsf{val}_2(v)| > 0$ is maximal, such a vertex exists since $\mathsf{val}_1 \neq \mathsf{val}_2$. For this v, $\mathsf{offset}(\mathsf{val}_1,(v,\sigma(v))) = \mathsf{offset}(\mathsf{val}_2,(v,\sigma(v)))$ entails $\mathsf{val}_1(v)-\mathsf{val}_2(v) = \lambda_{(v,\sigma(v))}(\mathsf{val}_1(\sigma(v))-\mathsf{val}_2(\sigma(v)))$. Using $\lambda_e \in [0,1)$, we get $|\mathsf{val}_1(\sigma(v))-\mathsf{val}_2(\sigma(v))| > |\mathsf{val}_1(v)-\mathsf{val}_2(v)| > 0$, which contradicts the maximality of v. Note that neither λ_e, nor the difference of $\mathsf{val}_1(\sigma(v))$ and $\mathsf{val}_2(\sigma(v))$ can be equal to 0 since $\mathsf{val}_1(v) \neq \mathsf{val}_2(v)$. Hence the contradiction.

We therefore have that $\mathsf{offset}(\mathsf{val}_1,(v,\sigma(v))) \neq \mathsf{offset}(\mathsf{val}_2,(v,\sigma(v)))$ holds for some $v \in V$. As the α_e are drawn independently, we can assume w.l.o.g. that $\alpha_{(v,\sigma(v))}$ is drawn last. There is at most one value $\alpha'_{(v,\sigma(v))}$ for which the condition

$$\sum_{v\in V} \mathsf{offset}'(\mathsf{val}_1,(v,\sigma(v))) \neq \sum_{v\in V} \mathsf{offset}'(\mathsf{val}_2,(v,\sigma(v)))$$

is not satisfied.

It therefore holds almost surely for all strategies that all base-induced valuations have a pairwise distinct image by the objective function associated to the strategy. This immediately implies that the game is improving. □

Thus, we can almost surely obtain sharpness by adding small noise to the weights, and almost surely make games improving by considering the offsets of the individual edges with a randomly chosen positive weight. This guarantees cheap progress for the case where there are no local improvements.

5.5 Mixing Pivoting on the Simplex and of the Objective

When using a simplex based technique to implement $\mathsf{LinearProgramming}$ (Line 5 of Algorithm 1), then the algorithm mixes three approaches that stepwise reduce the value of $f_\sigma(\mathsf{val})$:

1. The simplex algorithm updates the base, changing val (while retaining the objective function f_σ).
2. Local updates, who change the objective function f_σ (through updating σ) and retain val.
3. Non-local updates.

Non-local updates are more complex than the other two, and the correctness proofs make use of the non-existence of the other two improvements. For both reasons, it seems natural to take non-local updates as a last resort.

The other two updates, however, can be freely mixed, as they both bring down the value of $f_\sigma(\mathsf{val})$ by applying local changes. That the improvements from (1) are given preference in the algorithm is a choice made to keep the implementation of the algorithm for using linear programs open, allowing, for example, to use ellipsoid methods [20] or inner point methods [19] to keep this step tractable.

6 Discussion

There is widespread belief that mean payoff and discounted payoff games have two types of algorithmic solutions: value iteration [16, 21] and strategy improvement [25, 29, 6, 30, 33]. We have added a third method, which is structurally different and opens a new class of algorithms to attack these games. Moreover, our new symmetric approach has the same proximity to linear programming as strategy improvement algorithms, which is an indicator of efficiency.

Naturally, a fresh approach opens the door to much follow-up research. A first target for such research is the questions on how to arrange the selection of better strategies to obtain fewer updates, either proven on benchmarks, or theoretically in worst, average, or smoothed analysis. In particular, it would be interesting to establish non-trivial upper or lower bounds for various pivoting rules. Without such a study, a trivial bound for the proposed approach is provided by the number of strategies (exponential). Moreover, the lack of a benchmarking framework for existing algorithms prevents us from testing and compare an eventual implementation.

A second question is whether this method as whole can be turned into an inner point method. If so, this could be a first step on showing tractability of discounted payoff games – which would immediately extend to mean-payoff and parity games.

Acknowledgements

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 101032464. It was supported by the EPSRC through the projects EP/X017796/1 (Below the Branches of Universal Trees) and EP/X03688X/1 (TRUSTED: SecuriTy SummaRies for SecUre SofTwarE Development).

References

[1] L. de Alfaro, T.A. Henzinger & R. Majumdar (2001): *From Verification to Control: Dynamic Programs for Omega-Regular Objectives.* In: *Logic in Computer Science'01*, IEEE Computer Society, pp. 279–290. Available at `https://doi.org/10.1109/LICS.2001.932504`.

[2] R. Alur, T.A. Henzinger & O. Kupferman (2002): *Alternating-Time Temporal Logic.* *Journal of the ACM* 49(5), pp. 672–713. Available at `https://doi.org/10.1145/585265.585270`.

[3] M. Benerecetti, D. Dell'Erba & F. Mogavero (2016): *Improving Priority Promotion for Parity Games.* In: *Haifa Verification Conference16*, LNCS 10028, Springer, pp. 1–17. Available at `https://doi.org/10.1007/978-3-319-49052-6_8`.

[4] M. Benerecetti, D. Dell'Erba & F. Mogavero (2018): *A Delayed Promotion Policy for Parity Games.* *Information and Computation* 262(2), pp. 221–240. Available at `https://doi.org/10.1016/j.ic.2018.09.005`.

[5] M. Benerecetti, D. Dell'Erba & F. Mogavero (2020): *Solving Mean-Payoff Games via Quasi Dominions.* In: *Tools and Algorithms for the Construction and Analysis of Systems'20*, LNCS 12079, Springer, pp. 289–306. Available at `https://doi.org/10.1007/978-3-030-45237-7_18`.

[6] H. Björklund & S.G. Vorobyov (2007): *A Combinatorial Strongly Subexponential Strategy Improvement Algorithm for Mean-Payoff Games.* *Discrete Applied Mathematics* 155(2), pp. 210–229. Available at `https://doi.org/10.1016/j.dam.2006.04.029`.

[7] A. Browne, E.M. Clarke, S. Jha, D.E. Long & W.R. Marrero (1997): *An Improved Algorithm for the Evaluation of Fixpoint Expressions.* *Theoretical Computer Science* 178(1-2), pp. 237–255. Available at `https://doi.org/10.1016/S0304-3975(96)00228-9`.

[8] C. S. Calude, S. Jain, B. Khoussainov, W .Li & F .Stephan (2022): *Deciding Parity Games in Quasi-polynomial Time.* *SIAM Journal on Computing* 51(2), pp. 17–152. Available at `https://doi.org/10.1137/17M1145288`.

[9] Anne Condon (1993): *On Algorithms for Simple Stochastic Games.* In: *Advances in Computational Complexity Theory*, 13, American Mathematical Society, pp. 51–73. Available at `https://doi.org/10.1090/dimacs/013/04`.

[10] D. Dell'Erba & S. Schewe (2022): *Smaller progress measures and separating automata for parity games.* *Frontiers in Computer Science* 4. Available at `https://doi.org/10.3389/fcomp.2022.936903`.

[11] E.A. Emerson & C.S. Jutla (1991): *Tree Automata, muCalculus, and Determinacy.* In: *Foundation of Computer Science'91*, IEEE Computer Society, pp. 368–377. Available at `https://doi.org/10.1109/SFCS.1991.185392`.

[12] E.A. Emerson, C.S. Jutla & A.P. Sistla (1993): *On Model-Checking for Fragments of muCalculus.* In: *Computer Aided Verification'93*, LNCS 697, Springer, pp. 385–396. Available at `https://doi.org/10.1007/3-540-56922-7_32`.

[13] E.A. Emerson & C.-L. Lei (1986): *Efficient Model Checking in Fragments of the Propositional muCalculus.* In: *Logic in Computer Science'86*, IEEE Computer Society, pp. 267–278.

[14] J. Fearnley (2010): *Non-Oblivious Strategy Improvement.* In: *Logic for Programming Artificial Intelligence and Reasoning'10*, LNCS 6355, Springer, pp. 212–230. Available at `https://doi.org/10.1007/978-3-642-17511-4_13`.

[15] J. Fearnley, S. Jain, B. de Keijzer, S. Schewe, F. Stephan & D. Wojtczak (2019): *An Ordered Approach to Solving Parity Games in Quasi Polynomial Time and Quasi Linear Space.* *Software Tools for Technology Transfer* 21(3), pp. 325–349. Available at `https://doi.org/10.1007/s10009-019-00509-3`.

[16] N. Fijalkow, P. Gawrychowski & P. Ohlmann (2020): *Value Iteration Using Universal Graphs and the Complexity of Mean Payoff Games.* In: *Mathematical Foundations of Computer Science'20*, LIPIcs 170, Leibniz-Zentrum fuer Informatik, pp. 1–15. Available at `https://doi.org/10.4230/LIPIcs.MFCS.2020.34`.

[17] M. Jurdziński (1998): *Deciding the Winner in Parity Games is in UP $\cap$ co-UP.* *Information Processing Letters* 68(3), pp. 119–124. Available at `https://doi.org/10.1016/S0020-0190(98)00150-1`.

[18] M. Jurdziński & R. Lazic (2017): *Succinct Progress Measures for Solving Parity Games.* In: *Logic in Computer Science'17*, Association for Computing Machinery, pp. 1–9. Available at `https://doi.org/10.1109/LICS.2017.8005092`.

[19] N. Karmarkar (1984): *A new polynomial-time algorithm for linear programming.* In: *Symposium on Theory of Computing'84*, Association for Computing Machinery, pp. 302–311. Available at `https://doi.org/10.1007/BF02579150`.

[20] L. G. Khachian (1979): *A Polynomial Algorithm in Linear Programming.* *USSR Computational Mathematics and Mathematical Physics* 244, pp. 1093–1096. Available at `https://doi.org/10.1016/0041-5553(80)90061-0`.

[21] A. Kozachinskiy (2021): *Polyhedral Value Iteration for Discounted Games and Energy Games.* In: *Symposium on Discrete Algorithms'21*, SIAM, p. 600–616. Available at `https://doi.org/10.1137/1.9781611976465.37`.

[22] D. Kozen (1983): *Results on the Propositional muCalculus.* *Theoretical Computer Science* 27(3), pp. 333–354. Available at `https://doi.org/10.1016/0304-3975(82)90125-6`.

[23] K. Lehtinen & U. Boker (2020): *Register Games.* *Logical Methods in Computer Science* 16(2). Available at `https://doi.org/10.23638/LMCS-16(2:6)2020`.

[24] K. Lehtinen, P. Parys, S. Schewe & D. Wojtczak (20220): *A Recursive Approach to Solving Parity Games in Quasipolynomial Time.* *Logical Methods in Computer Science* 18(1). Available at `https://doi.org/10.46298/lmcs-18(1:8)2022`.

[25] W. Ludwig (1995): *A Subexponential Randomized Algorithm for the Simple Stochastic Game Problem.* *Information and Computation* 117(1), pp. 151–155. Available at `https://doi.org/10.1006/inco.1995.1035`.

[26] R. McNaughton (1993): *Infinite Games Played on Finite Graphs.* *Annals of Pure and Applied Logic* 65, pp. 149–184. Available at `https://doi.org/10.1016/0168-0072(93)90036-D`.

[27] J. Obdrzálek (2003): *Fast Mu-Calculus Model Checking when Tree-Width Is Bounded.* In: *Computer Aided Verification'03*, LNCS 2725, Springer, pp. 80–92. Available at `https://doi.org/10.1007/978-3-540-45069-6_7`.

[28] N. Piterman (2006): *From Nondeterministic Buchi and Streett Automata to Deterministic Parity Automata.* In: Logic in Computer Science'06, IEEE Computer Society, pp. 255–264. Available at `https://doi.org/10.2168/LMCS-3(3:5)2007`.

[29] A. Puri (1995): *Theory of Hybrid Systems and Discrete Event Systems.* Ph.D. thesis, University of California, Berkeley, USA. Available at `https://www2.eecs.berkeley.edu/Pubs/TechRpts/1995/ERL-95-113.pdf`.

[30] S. Schewe (2008): *An Optimal Strategy Improvement Algorithm for Solving Parity and Payoff Games.* In: Computer Science Logic'08, LNCS 5213, Springer, pp. 369–384. Available at `https://doi.org/10.1007/978-3-540-87531-4_27`.

[31] S. Schewe & B. Finkbeiner (2006): *Satisfiability and Finite Model Property for the Alternating-Time muCalculus.* In: Computer Science Logic'06, LNCS 6247, Springer, pp. 591–605. Available at `https://doi.org/10.1007/11874683_39`.

[32] S. Schewe & B. Finkbeiner (2006): *Synthesis of Asynchronous Systems.* In: Symposium on Logic-based Program Synthesis and Transformation'06, LNCS 4407, Springer, pp. 127–142. Available at `https://doi.org/10.1007/978-3-540-71410-1_10`.

[33] S. Schewe, A. Trivedi & T. Varghese (2015): *Symmetric Strategy Improvement.* In: International Colloquium on Automata, Languages, and Programming'15, LNCS 9135, Springer, pp. 388–400. Available at `https://doi.org/10.1007/978-3-662-47666-6_31`.

[34] M.Y. Vardi (1998): *Reasoning about The Past with Two-Way Automata.* In: International Colloquium on Automata, Languages, and Programming'98, LNCS 1443, Springer, pp. 628–641. Available at `https://doi.org/10.1007/BFb0055090`.

[35] J. Vöge & M. Jurdziński (2000): *A Discrete Strategy Improvement Algorithm for Solving Parity Games.* In: Computer Aided Verification'00, LNCS 1855, Springer, pp. 202–215. Available at `https://doi.org/10.1007/10722167_18`.

[36] T. Wilke (2001): *Alternating Tree Automata, Parity Games, and Modal muCalculus.* Bulletin of the Belgian Mathematical Society 8(2), pp. 359–391. Available at `https://doi.org/10.36045/bbms/1102714178`.

[37] W. Zielonka (1998): *Infinite Games on Finitely Coloured Graphs with Applications to Automata on Infinite Trees.* Theoretical Computer Science 200(1-2), pp. 135–183. Available at `https://doi.org/10.1016/S0304-3975(98)00009-7`.

[38] U. Zwick & M. Paterson (1996): *The Complexity of Mean Payoff Games on Graphs.* Theoretical Computer Science 158(1-2), pp. 343–359. Available at `https://doi.org/10.1016/0304-3975(95)00188-3`.

Strategies Resilient to Delay: Games under Delayed Control vs. Delay Games

Martin Fränzle
Carl v. Ossietzky Universität
Oldenburg, Germany
martin.fraenzle@uol.de

Sarah Winter
Université libre de Bruxelles
Brussels, Belgium
swinter@ulb.ac.be

Martin Zimmermann
Aalborg University
Aalborg, Denmark
mzi@cs.aau.dk

We compare games under delayed control and delay games, two types of infinite games modelling asynchronicity in reactive synthesis. Our main result, the interreducibility of the existence of sure winning strategies for the protagonist, allows to transfer known complexity results and bounds on the delay from delay games to games under delayed control, for which no such results had been known. We furthermore analyze existence of randomized strategies that win almost surely, where this correspondence between the two types of games breaks down.

1 Introduction

Two-player zero-sum games of infinite duration are a standard model for the synthesis of reactive controllers, i.e., correct-by-construction controllers that satisfy their specification even in the presence of a malicious environment. In such games, the interaction between the controller and the environment is captured by the rules of the game and the specification on the controller induces the winning condition of the game. Then, computing a correct controller boils down to computing a winning strategy.

Often, it is convenient to express the rules in terms of a graph capturing the state-space such that moves correspond to transitions between these states. The interaction between the controller and the environment then corresponds to a path through the graph and the winning condition is a language of such paths, containing those that correspond to interactions that satisfy the specification on the controller.

In other settings, it is more convenient to consider a slightly more abstract setting without game graphs, so-called Gale-Stewart games [4]. In such games, the players alternatingly pick a sequence of letters, thereby constructing an infinite word. The winning condition is a language over infinite words, containing the winning words for one player. To capture the synthesis problem, the winning condition has to encode both the specification on the controller as well as the rules of interaction. It is straightforward to transform a graph-based game into a Gale-Stewart game and a Gale-Stewart game into a graph-based game such that the existence of winning strategies for both players is preserved.

In the most basic setting of synthesis, both the controller and the environment are fully informed about the current state of the game (complete information). However, this scenario is not always realistic. Thus, much effort has been poured into studying games under incomplete information where the players are only partially informed about the current state of the game. Here, we are concerned with a special type of partial information designed to capture delays in perception and action. Such delays either render the most recent moves of the opponent invisible to a player or induce a time lag between the selection and the implementation of an own move, respectively.

As a motivating example, consider the domain of cooperative driving: Here, the exchange of information between cars is limited (and therefore delayed) by communication protocols that have to manage the available bandwidth to transfer information between cars. Other delaying factors include, e.g., complex

A. Achilleos and D. Della Monica (Eds.): Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification (GandALF 2023).
EPTCS 390, 2023, pp. 220–235, doi:10.4204/EPTCS.390.14

signal processing chains based on computer vision to detect the locations of obstacles. Thus, decisions have to be made based on incomplete information, which only arrives after some delay.

Games under Delayed Control. Chen et al. [2] introduced (graph) games under delayed control to capture this type of incomplete information. Intuitively, assume the players so far have constructed a finite path $v_0 \cdots v_k$ through the graph. Then, the controller has to base her decision on a visible proper prefix $v_0 \cdots v_{k-\delta}$, where δ is the amount of delay. Hence, the suffix $v_{k-\delta+1} \cdots v_k$ is not yet available to base the decision on, although the decision to be made is to be applied at the last state v_k in the sequence.

They showed that solving games under delayed control with safety conditions and with respect to a given delay is decidable: They presented two algorithms, an exponential one based on a reduction to delay-free safety games using a queue of length δ, and a more practical incremental algorithm synthesizing a series of controllers handling increasing delays and reducing game-graph size in between. They showed that even a naïve implementation of this algorithm outperforms the reduction-based one, even when the latter is used with state-of-the-art solvers for delay-free games. However, the exact complexity of the incremental algorithm and that of solving games under delayed control remained open.

Note that asking whether there is some delay δ that allows controller to win reduces to solving standard, i.e., delay-free games, as they correspond to the case $\delta = 0$. The reason is monotonicity in the delay: if the controller can win for delay δ then also for any $\delta' < \delta$. More interesting is the question whether controller wins with respect to every possible delay. Chen et al. conjectured that there is some exponential δ such that if the controller wins under delay δ, then also under every δ'.

Delay Games. There is also a variant of Gale-Stewart games modelling delayed interaction between the players [7]. Here, the player representing the environment (often called Player I) has to provide a lookahead on her moves, i.e., the player representing the controller (accordingly called Player O) has access to the first $n+k$ letters picked by Player I when picking her n-th letter. So, k is the amount of lookahead that Player I has to grant Player O. Note that the lookahead benefits Player O (representing the controller) while the delay in a game under delayed control disadvantages the controller.

Only three years after the seminal Büchi-Landweber theorem showing that delay-free games with ω-regular winning conditions are decidable [1], Hosch and Landweber showed that it is decidable whether there is a k such that Player O wins a given Gale-Stewart game with lookahead k [7]. Forty years later, Holtmann, Kaiser, and Thomas [6] revisited these games (and dubbed them delay games). They proved that if Player O wins a delay game then she wins it already with at most doubly-exponential lookahead (in the size of a given deterministic parity automaton recognizing the winning condition). Thus, unbounded lookahead does not offer any advantage over doubly-exponential lookahead in games with ω-regular winning conditions. Furthermore, they presented an algorithm with doubly-exponential running time solving delay games with ω-regular winning conditions conditions, i.e., determining whether there exists a k such that Player O wins a given delay game (with its winning condition again given by a deterministic parity automaton) with lookahead k.

Both upper bounds were improved and matching lower bounds were later proven by Klein and Zimmermann [9]: Solving delay games is EXPTIME-complete and exponential lookahead is both necessary to win some games and sufficient to win all games that can be won. Both lower bounds already hold for winning conditions specified by deterministic safety automata while the upper bounds hold for deterministic parity automata. The special case of solving games with conditions given as reachability automata is PSPACE-complete, but exponential lookahead is still necessary and sufficient. Thus, there are tight complexity results for delay games, unlike for games under delayed control.

Our Contributions. In this work, we exhibit a tight relation between controller in a game under delayed control and Player I in a delay game (recall that these are the players that are disadvantaged by delay and lookahead, respectively). Note that winning conditions in games under delayed control are always given from the perspective of controller (i.e., she has to avoid unsafe states in a safety game) while winning conditions in delay games are always given from the perspective of Player O. Hence, as we relate controller and Player I, we always have to complement winning conditions.

More precisely, we show that one can transform a safety game under delayed control in polynomial time into a delay game with a reachability condition for Player O (i.e., with a safety condition for Player I) such that controller wins the game under delayed control with delay δ if and only if Player I wins the resulting delay game with lookahead of size $\frac{\delta}{2}$. Dually, we show that one can transform a delay game with safety condition for Player I in polynomial time into a reachability game under delayed control such that Player I wins the delay game with lookahead of size δ if and only if controller wins the resulting game under delayed control with delay 2δ. Thus, we can transfer both upper and lower bound results on complexity and on (necessary and sufficient) lookahead from delay games to delayed control. In particular, determining whether controller wins a given safety game under delayed control for every possible delay is PSPACE-complete. Our reductions also prove the conjecture by Chen et al. on the delays that allow controller to win such games. Furthermore, we generalize our translation from games with safety conditions to games with parity conditions and games with winning conditions given by formulas of Linear Temporal Logic (LTL) [12], again allowing us to transfer known results for delay games to games under delayed control.

Note that we have only claimed that the existence of winning strategies for the controller in the game under delayed control and Player I in the delay game coincides. This is no accident! In fact, the analogous result for relating environment and Player O fails. This follows immediately from the fact that delay games are determined while games under delayed control are undetermined, even with safety conditions. The reason is that the latter games are truly incomplete information games (which are typically undetermined) while delay games are perfect information games.

We conclude by a detailed comparison between environment and Player O in both the setting with deterministic as well as in the setting with randomized strategies. The latter setting increases power for both the controller and the environment, making them win (almost surely) games under delayed control that remain undetermined in the deterministic setting, but it also breaks the correspondence between controller and Player I observed in the deterministic setting: there are games that controller wins almost surely while Player I surely looses them.

All proofs which are omitted due to space restrictions can be found in the full version [3].

2 Preliminaries

We denote the non-negative integers by $\mathbb{N}$. An alphabet Σ is a non-empty finite set of letters. A word over Σ is a finite or infinite sequence of letters of Σ: The set of finite words (non-empty finite words, infinite words) over Σ is denoted by Σ^* (Σ^+, Σ^ω). The empty word is denoted by ε, the length of a finite word w is denoted by $|w|$. Given two infinite words $\alpha \in (\Sigma_0)^\omega$ and $\beta \in (\Sigma_1)^\omega$, we define $\binom{\alpha}{\beta} = \binom{\alpha(0)}{\beta(0)}\binom{\alpha(1)}{\beta(1)}\binom{\alpha(2)}{\beta(2)}\cdots \in (\Sigma_0 \times \Sigma_1)^\omega$.

2.1 Games under Delayed Control

Games under delayed control are played between two players, controller and environment. For pronomial convenience [11], we refer to controller as she and environment as he.

A game $\mathcal{G} = (S, s_0, S_c, S_e, \Sigma_c, \Sigma_e, \rightarrow, \mathrm{Win})$ consists of a finite set S of states partitioned into the states $S_c \subseteq S$ of the controller and the states $S_e \subseteq S$ of the environment, an initial state $s_0 \in S_c$, the sets of actions Σ_c for the controller and Σ_e for the environment, a transition function $\rightarrow\colon (S_c \times \Sigma_c) \cup (S_e \times \Sigma_e) \rightarrow S$ such that $s \in S_c$ and $\sigma \in \Sigma_c$ implies $\rightarrow(s,\sigma) \in S_e$ and vice versa, and a winning condition $\mathrm{Win} \subseteq S^\omega$. We write $s \xrightarrow{\sigma} s'$ as shorthand for $s' = \rightarrow(s,\sigma)$.

A play in $\mathscr{G}$ is an infinite sequence $\pi = \pi_0\sigma_0\pi_1\sigma_1\pi_2\sigma_2\cdots$ satisfying $\pi_0 = s_0$ and $\pi_n \xrightarrow{\sigma_n} \pi_{n+1}$ for all $n \geq 0$. We say that controller wins π if $\pi_0\pi_1\pi_2\cdots \in \mathrm{Win}$; otherwise, we say that environment wins π. The play prefix of π of length n is defined as $\pi[n] = \pi_0\sigma_0\cdots\sigma_{n-1}\pi_n$, i.e., n is the number of actions (equivalently, the number of transitions). We denote by $\mathrm{Pref}(\mathscr{G})$ the set of play prefixes of all plays in $\mathscr{G}$, which is partitioned into the sets $\mathrm{Pref}_c(\mathscr{G})$ and $\mathrm{Pref}_e(\mathscr{G})$ of play prefixes ending in S_c and S_e, respectively. Due to our alternation assumption, play prefixes of even (odd) length are in $\mathrm{Pref}_c(\mathscr{G})$ ($\mathrm{Pref}_e(\mathscr{G})$).

Fix some even $\delta \geq 0$. A strategy for the controller in $\mathscr{G}$ under delay δ is a pair (α,τ_c) where $\alpha \in (\Sigma_c)^{\frac{\delta}{2}}$ and $\tau_c\colon \mathrm{Pref}_c(\mathscr{G}) \rightarrow \Sigma_c$ maps play prefixes ending in S_c to actions of the controller. A play $\pi_0\sigma_0\pi_1\sigma_1\pi_2\sigma_2\cdots$ is consistent with (α,τ_c) if $\sigma_0\sigma_2\cdots\sigma_{\delta-4}\sigma_{\delta-2} = \alpha$ and $\sigma_{2n} = \tau_c(\pi[2n-\delta])$ for all $2n > \delta-2$, i.e., controller has access to environment's actions with a delay of δ. In particular, her first $\frac{\delta}{2}+1$ actions are independent of environment's actions and, in general, her n-th action σ_{2n} only depends on the actions $\sigma_1,\ldots,\sigma_{(2n-\delta)-1}$ picked by environment, but not on the actions $\sigma_{(2n-\delta)+1},\ldots,\sigma_{2n-1}$. The strategy (α,τ_c) is winning under delay δ if every play that is consistent with it is winning for controller. Controller wins $\mathscr{G}$ under delay δ if she has a winning strategy under delay δ for $\mathscr{G}$.

Remark 1.

1. *The notion of winning strategy for controller under delay* 0 *is the classical one for delay-free games (cf. [5]).*
2. *If controller wins $\mathscr{G}$ under delay δ, then also under every delay $\delta' < \delta$ [2].*

A strategy for environment is a mapping $\tau_e\colon \mathrm{Pref}_e(\mathscr{G}) \rightarrow \Sigma_e$. A play $\pi_0\sigma_0\pi_1\sigma_1\pi_2\sigma_2\cdots$ is consistent with τ_e if $\sigma_{2n+1} = \tau_e(\pi_0\sigma_0\cdots\sigma_{2n-1}\pi_{2n+1})$ for all $n \geq 0$, i.e., environment has access to the full play prefix when picking his next action. The strategy τ_e is winning, if every play that is consistent with it is winning for the environment (i.e., the sequence of states is not in Win). Further, we say that environment wins $\mathscr{G}$, if he has a winning strategy for $\mathscr{G}$. Note that the two definitions of strategies are in general not dual, e.g., the one for environment is not defined with respect to a delay δ.

Remark 2. *The notion of winning strategy for environment is the classical one for delay-free games (cf. [5]).*

We say that a game under delayed control $\mathscr{G}$ is determined under delay δ, if either controller wins $\mathscr{G}$ under delay δ or environment wins $\mathscr{G}$. Let us stress that determinacy is defined with respect to some fixed δ and that $\mathscr{G}$ may be determined for some δ, but undetermined for some other δ' (due to the non-dual definition of strategies). Remark 9 shows an undetermined safety (!) game under delayed control.

Example 1. *Consider the game $\mathscr{G} = (S, s_I, S_c, S_e, \Sigma_c, \Sigma_e, \rightarrow, \mathrm{Win})$ depicted in Fig. 1 where* Win *contains all plays that do not visit the black vertex. Note that this is a safety condition. In particular, if controller does not pick action b at c_2 and does not pick action a at c_3, then the vertex e_3 is never reached. This is straightforward without delay, but we claim that controller can also win $\mathscr{G}$ under delay* 2.

To gain some intuition, consider a play prefix $\pi_0\sigma_0\pi_1\cdots\pi_{n-1}\sigma_{n-1}\pi_n$ with $n \geq 4$ and $\pi_n \in S_c$. Then, controller has to pick an action σ_n to continue the prefix. However, due to the delayed control, she has to do so based on the prefix $\pi_0\sigma_0\pi_1\cdots\pi_{n-3}\sigma_{n-3}\pi_{n-2}$.

If π_{n-2} is c_2, then π_n is either c_3 or c_1. Hence, picking $\sigma_n = b$ is the only safe choice. Dually, if π_{n-2} is c_3, then π_n is either c_2 or c_1. Hence, picking $\sigma_n = a$ is the only safe choice.

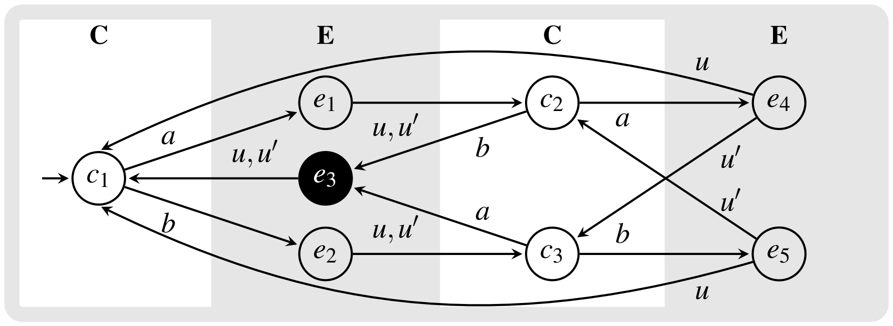


Figure 1: The game for Example 1. Controller wins all plays that never visit the black vertex. Note that we have $\Sigma_c = \{a, b\}$ and $\Sigma_e = \{u, u'\}$.

Finally, assume π_{n-2} is c_1. Then, π_n is either c_2 or c_3. In the former case, picking $\sigma_n = a$ is the only safe choice, in the latter case, picking $\sigma_n = b$ is the only safe choice. So, controller needs to distinguish these two cases, although she has no access to π_n.

But she can do so by inspecting π_{n-3} (which she has access to): As a predecessor of $\pi_{n-2} = c_1$, it can either be e_4, e_5, or e_3. In the latter case, the play is already losing. Thus, we disregard this case, as we construct a winning strategy. So, assume we have $\pi_{n-3} = e_4$ (the case $\pi_{n-3} = e_5$ is dual). Then, we must have $\pi_{n-4} = c_2$ (the only predecessor of e_4) and, by our analysis of the safe moves above, controller must have picked $\sigma_{n-2} = b$ (based, due to delay, on the prefix ending in $\sigma_{n-4} = c_2$). From this we can conclude $\pi_{n-1} = e_2$ and thus $\pi_n = c_3$ (the only successor of e_2). Thus, she can safely pick $\sigma_n = b$.

This intuition, and the necessary initialization, is implemented by the strategy (α, τ_c) with $\alpha = a$ and

$$\tau_c(\pi_0\sigma_0\pi_1\cdots\pi_{n-3}\sigma_{n-3}\pi_{n-2}) = \begin{cases} a & n = 2 \text{ and } \pi_0 = c_1, \\ b & n > 2,\ \pi_{n-2} = c_1, \text{ and } \pi_{n-3} = e_4, \\ a & n > 2,\ \pi_{n-2} = c_1, \text{ and } \pi_{n-3} = e_5, \\ b & \pi_{n-2} = c_2, \\ a & \pi_{n-2} = c_3. \end{cases}$$

An induction over the play length shows that (α, τ_c) is winning for controller under delay 2.

Remark 3. *Our definition of games under delayed control differs in three aspects from the original definition of Chen et al. [2].*

- *We allow arbitrary winning conditions while Chen et al. focused on safety conditions.*
- *The original definition allows nondeterministic strategies (a strategy that returns a nonempty set of actions, each one of which can be taken), while we restrict ourselves here to deterministic strategies (a strategy that returns a single action to be taken). The motivation for their use of nondeterministic strategies is the fact that they can be refined if additional constraints are imposed, which Chen et al.'s algorithm computing a winning strategy relies on.*

  Here, on the other hand, we are just interested in the existence of winning strategies. In this context, it is sufficient to consider deterministic strategies, as controller has a nondeterministic winning strategy if and only if she has a deterministic winning strategy. Also, strategies in delay games are deterministic, so the transformation between games under delayed control and delay games can be formulated more naturally for deterministic strategies.

- *The original definition also allowed odd delays δ while we only allow even delays. As we will see in Section 3, the transformation of games under delayed control to delay games is naturally formulated for even delays. This choice also simplifies definitions, as accounting for odd delays imposes an additional notational burden.*

2.2 Delay Games

Delay games are played between two players, Player I (she) and Player O (he). A delay game $\Gamma_k(L)$ (with constant lookahead) consists of a lookahead $k \in \mathbb{N}$ and a winning condition $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ for some alphabets Σ_I and Σ_O. Such a game is played in rounds $n = 0, 1, 2, \dots$ as follows: in round 0, first Player I picks a word $x_0 \in \Sigma_I^{k+1}$, then Player O picks a letter $y_0 \in \Sigma_O$. In round $n > 0$, Player I picks a letter $x_n \in \Sigma_I$, then Player O picks a letter $y_n \in \Sigma_O$. Player O wins a play $(x_0, y_0)(x_1, y_1)(x_2, y_2)\cdots$ if the outcome $\binom{x_0 x_1 x_2 \cdots}{y_0 y_1 y_2 \cdots}$ is in L; otherwise, Player I wins.

A strategy for Player I in $\Gamma_k(L)$ is a mapping $\tau_I \colon \Sigma_O^* \to \Sigma_I^*$ satisfying $|\tau_I(\varepsilon)| = k+1$ and $|\tau_I(w)| = 1$ for all $w \in \Sigma_O^+$. A strategy for Player O is a mapping $\tau_O \colon \Sigma_I^+ \to \Sigma_O$. A play $(x_0, y_0)(x_1, y_1)(x_2, y_2)\cdots$ is consistent with τ_I if $x_n = \tau_I(y_0 \cdots y_{n-1})$ for all $n \geq 0$, and it is consistent with τ_O if $y_n = \tau_O(x_0 \cdots x_n)$ for all $n \geq 0$. So, strategies are dual in delay games, i.e., Player I has to grant some lookahead on her moves that Player O has access to. A strategy for Player $P \in \{I, O\}$ is winning, if every play that is consistent with the strategy is won by Player P. We say that Player $P \in \{I, O\}$ wins a game $\Gamma_k(L)$ if Player P has a winning strategy in $\Gamma_k(L)$.

Remark 4.

- *If Player O wins $\Gamma_k(L)$, then he also wins $\Gamma_{k'}(L)$ for every $k' > k$.*
- *If Player I wins $\Gamma_k(L)$, then she also wins $\Gamma_{k'}(L)$ for every $k' < k$.*

Unlike games under delayed control, delay games with Borel winning conditions are determined [9], i.e., each delay game $\Gamma_k(L)$ with Borel L and fixed k is won by one of the players.

Example 2. *Consider $L = \left\{ \binom{a_0}{b_0}\binom{a_1}{b_1}\binom{a_2}{b_2} \cdots \mid b_0 \notin \{a_0, a_1, a_2\} \right\}$ over the alphabets $\Sigma_I = \Sigma_O = \{1, 2, 3, 4\}$.*

Player I wins $\Gamma_k(L)$ for $k = 1$ with the following strategy τ_I: $\tau_I(\varepsilon) = 12$ and $\tau_I(b_0) = b_0$, and $\tau_I(w)$ arbitrary for all $w \in \Sigma_O^+$ with $|w| > 1$: In round 0, after Player I has picked $a_0 a_1 = 12$, Player O has to pick some b_0. In order to not loose immediately, he has to pick $b_0 \notin \{1, 2\}$. Then, in round 1, Player I picks $a_2 = b_0$ and thereby ensures $b_0 \in \{a_0, a_1, a_2\}$. Hence, the play is not won by Player O (it's outcome is not in L), therefore it is winning for Player I.

However, Player O wins $\Gamma_k(L)$ for $k = 2$ with the following strategy τ_O: $\tau_O(a_0 a_1 a_2)$ is a letter in the nonempty set $\Sigma_O \setminus \{a_0, a_1, a_2\}$ and $\tau_O(w)$ arbitrary for all $w \in \Sigma_I^*$ with $|w| \neq 3$. In round 0, after Player I has picked $a_0 a_1 a_2$, Player O picks $b_0 \notin \{a_0, a_1, a_2\}$ and thus ensures that the outcome is in L.

Remark 5. *We restrict ourselves here to the setting of constant lookahead, i.e., in a delay game $\Gamma_k(L)$ in round n when Player O picks her n-th letter, Player I has already picked $k + n + 1$ letters (note that we start in round 0 with the zeroth letter). Delay games have also been studied with respect to growing lookahead, i.e., the lookahead increases during a play [6]. However, it is known that constant lookahead is sufficient for all ω-regular winning conditions: if Player O wins for any lookahead (no matter how fast it is growing), then she also wins with respect to constant lookahead, which can even be bounded exponentially in the size of a deterministic parity automaton recognizing the winning condition [9]. Stated differently, growing lookahead does not allow to win any more games than constant lookahead. Finally, the setting of constant lookahead in delay games considered here is the natural counterpart to games under delayed control, where the delay is fixed during a play.*

2.3 ω-Automata

A deterministic reachability automaton $\mathscr{A} = (Q, \Sigma, q_I, \delta_{\mathscr{A}}, F)$ consists of a finite set Q of states containing the initial state $q_I \in Q$ and the set of accepting states $F \subseteq Q$, an alphabet Σ, and a transition function $\delta_{\mathscr{A}} \colon Q \times \Sigma \to Q$. The size of $\mathscr{A}$ is defined as $|\mathscr{A}| = |Q|$. Let $w = w_0 w_1 w_2 \cdots \in \Sigma^\omega$. The run of $\mathscr{A}$ on w is the sequence $q_0 q_1 q_2 \cdots$ such that $q_0 = q_I$ and $q_{n+1} = \delta_{\mathscr{A}}(q_n, w_n)$ for all $n \geq 0$. A run $q_0 q_1 q_2 \cdots$ is (reachability) accepting if $q_n \in F$ for some $n \geq 0$. The language (reachability) recognized by $\mathscr{A}$, denoted by $L(\mathscr{A})$, is the set of infinite words over Σ such that the run of $\mathscr{A}$ on w is (reachability) accepting.

A deterministic safety automaton has the form $\mathscr{A} = (Q, \Sigma, q_I, \delta_{\mathscr{A}}, U)$ where $Q, \Sigma, q_I, \delta_{\mathscr{A}}$ are as in a deterministic reachability automaton and where $U \subseteq Q$ is a set of unsafe states. The notions of size and runs are defined as for reachability automata, too. A run $q_0 q_1 q_2 \cdots$ is (safety) accepting if $q_n \notin U$ for all $n \geq 0$. The language (safety) recognized by $\mathscr{A}$, again denoted by $L(\mathscr{A})$, is the set of infinite words over Σ such that the run of $\mathscr{A}$ on w is (safety) accepting.

A deterministic parity automaton has the form $\mathscr{A} = (Q, \Sigma, q_I, \delta_{\mathscr{A}}, \Omega)$ where $Q, \Sigma, q_I, \delta_{\mathscr{A}}$ are as in a deterministic reachability automaton and where $\Omega \colon Q \to \mathbb{N}$ is a coloring of the states. The notions of size and runs are defined as for reachability automata, too. A run $q_0 q_1 q_2 \cdots$ is (parity) accepting if the maximal color appearing infinitely often in the sequence $\Omega(q_0)\Omega(q_1)\Omega(q_2)\cdots$ is even. The language (parity) recognized by $\mathscr{A}$, again denoted by $L(\mathscr{A})$, is the set of infinite words over Σ such that the run of $\mathscr{A}$ on w is (parity) accepting.

Reachability and safety automata are dual while parity automata are self-dual.

Remark 6.

1. *Let $\mathscr{A} = (Q, \Sigma, q_I, \delta_{\mathscr{A}}, F)$ be a deterministic reachability automaton and let $\overline{\mathscr{A}}$ be the deterministic safety automaton $(Q, \Sigma, q_I, \delta_{\mathscr{A}}, Q \setminus F)$. Then, $L(\overline{\mathscr{A}}) = \overline{L(\mathscr{A})}$.*

2. *Let $\mathscr{A} = (Q, \Sigma, q_I, \delta_{\mathscr{A}}, \Omega)$ be a deterministic parity automaton and let $\overline{\mathscr{A}}$ be the deterministic parity automaton $(Q, \Sigma, q_I, \delta_{\mathscr{A}}, q \mapsto \Omega(q) + 1)$. Then, $L(\overline{\mathscr{A}}) = \overline{L(\mathscr{A})}$.*

3 From Games under Delayed Control to Delay Games and Back

In this section, we exhibit a tight correspondence between controller in games under delayed control and Player I in delay games. Recall that in a game under delayed control, it is the controller whose control is delayed, i.e., she is at a disadvantage as she only gets delayed access to the action picked by environment. In a delay game, it is Player I who is at a disadvantage as she has to grant a lookahead on her moves to Player O. Thus, when simulating a game under delayed control by a delay game, it is natural to let Player I take the role of controller and let Player O take the role of environment. Also recall that the winning condition Win in a game under delayed control is formulated from controller's point-of-view: the winning condition requires her to enforce a play in Win. On the other hand, the winning condition L of a delay game is formulated from the point-of-view of Player O: Player O has to enforce a play whose outcome is in L. Thus, as Player I takes the role of controller, we need to complement the winning condition to reflect this change in perspective: The set of winning outcomes for Player I in the simulating delay game is the complement of Win.

In the remainder of this section, we show how to simulate a game under delayed control by a delay game and then the converse, i.e., how to simulate a delay game by a game under delayed control.

Transformation 1. *First, we transform a game under delayed control into a delay game. In the resulting delay game, the players simulate a play in the game under delayed control by picking actions,* *which uniquely induce such a play. To formalize this, we need to introduce some notation. Fix a game $\mathscr{G} = (S, s_0, S_c, S_e, \Sigma_c, \Sigma_e, \rightarrow, \mathrm{Win})$. Note that a sequence $\sigma_0\sigma_1\sigma_2\cdots \in (\Sigma_c\Sigma_e)^\omega$ induces a unique play* $\mathrm{play}(\sigma_0\sigma_1\sigma_2\cdots) = \pi_0\sigma_0\pi_1\sigma_1\pi_2\sigma_2\cdots$ *in $\mathscr{G}$ which is defined as follows: $\pi_0 = s_0$ and $\pi_{n+1} = \rightarrow(\pi_n, \sigma_n)$ for all $n \geq 0$. Likewise, a finite sequence $\sigma_0\sigma_1\cdots\sigma_n \in (\Sigma_c\Sigma_e)^*(\Sigma_c + \varepsilon)$ induces a unique play prefix* $\mathrm{play}(\sigma_0\sigma_1\cdots\sigma_n)$ *which is defined analogously.*

Now, we define the language $L(\mathscr{G}) \subseteq (\Sigma_c \times \Sigma_e)^\omega$ such that $\binom{\sigma_0}{\sigma_1}\binom{\sigma_2}{\sigma_3}\binom{\sigma_4}{\sigma_5}\cdots \in L(\mathscr{G})$ if and only if $\mathrm{play}(\sigma_0\sigma_1\sigma_2\cdots)$ *is winning for controller.*

Now, we prove the correspondence between $\mathscr{G}$ and $\Gamma_k(\overline{L(\mathscr{G})})$. The winning condition of the delay game is the complement of $L(\mathscr{G})$, which implements the switch of perspective described above.

Lemma 1. *Let $\mathscr{G}$ be a game and $\delta \geq 0$ even. Controller wins $\mathscr{G}$ under delay δ if and only if Player I wins $\Gamma_k(\overline{L(\mathscr{G})})$ for $k = \frac{\delta}{2}$.*

Now, we consider the converse and transform a delay game into a game under delayed control.

Transformation 2. *Fix a delay game $\Gamma_k(L)$. We construct a game under delayed control to simulate $\Gamma_k(L)$ as follows: The actions of controller are the letters in Σ_I, and the actions of environment are the letters in Σ_O. Thus, by picking actions, controller and environment construct the outcome of a play of $\Gamma_k(L)$. As winning conditions of games under delayed control only refer to states visited by a play, but not the actions picked by the players, we reflect the action picked by a player in the state reached by picking that action. Here, we have to require without loss of generality that Σ_I and Σ_O are disjoint.*

Formally, we define $\mathscr{G}(L) = (S, s_0, S_c, S_e, \Sigma_c, \Sigma_e, \rightarrow, \mathrm{Win})$ with $S = S_c \cup S_e$, $S_c = \{s_0\} \cup \Sigma_O$, $S_e = \Sigma_I$, $\Sigma_c = \Sigma_I$, $\Sigma_e = \Sigma_O$, $\rightarrow(s,a) = a$ for all $s \in S_c$ and $a \in \Sigma_I$, and $\rightarrow(s,b) = b$ for all $s \in S_e$ and $b \in \Sigma_O$. Finally, we define $\mathrm{Win} = \{s_0s_1s_2\cdots \mid \binom{s_0}{s_1}\binom{s_2}{s_3}\binom{s_4}{s_5}\cdots \in L\}$.

The following remark states that the two transformations are inverses of each other, which simplifies the proof of correctness of the second transformation. It follows by a careful inspection of the definitions.

Remark 7. *Let $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$. Then, $L = L(\mathscr{G}(L))$.*

Now, we show that the second transformation is correct, again using complementation to implement the perspective switch.

Lemma 2. *Let $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ and $k \geq 0$. Player I wins $\Gamma_k(L)$ if and only if controller wins $\mathscr{G}(\overline{L})$ under delay $2k$.*

4 Results

Lemma 1 and Lemma 2 allow us to transfer results from delay games to games under delayed control. Due to the definitions of strategies in games under delayed control not being dual, we consider both players independently, controller in Section 4.1 and environment in Section 4.2.

Recall that delay that allows controller to win satisfies a monotonicity property (see Remark 1.2): if controller wins a game under delay δ, then also under every delay $\delta' < \delta$. Thus, the set of delays for which controller wins is downward-closed, i.e., it is either a finite set $\{0, 2, 4 \ldots, \delta_{\max}\}$ or it is equal to the set $2\mathbb{N}$ of even numbers. In the following, we study the complexity of determining whether controller wins under all possible delays, whether she wins under a given delay, and determine bounds on $\delta_{\max}$.

Note that winning for environment is independent of delay and boils down to the classical notion of winning delay-free games [5], which is a well-studied problem. Hence, we disregard this problem. However, we do discuss the relation between environment in a game under delayed control and Player O in the simulating delay game constructed in the previous section.

4.1 Controller's View

Before we present our results, we need to specify how to measure the size of games and delay games, especially how winning conditions are represented (recall that, so far, they are just ω-languages). In the following, we only consider ω-regular winning conditions specified by ω-automata (see Section 2.3) or formulas of Linear Temporal Logic (LTL) [12], which subsume the typical specification languages for winning conditions. Hence, the size of a game $(S, s_0, S_c, S_e, \Sigma_c, \Sigma_e, \rightarrow, \mathrm{Win})$ under delayed control is given by the sum $|S| + |\Sigma_c| + |\Sigma_e| + |\mathrm{Win}|$, where $|\mathrm{Win}|$ is the size of an automaton or LTL formula (measured in the number of distinct subformulas) representing Win. Analogously, for a delay game $\Gamma_k(L)$, we define the size of L as the size of an automaton or LTL formula (measured in the number of distinct subformulas) representing L. The bound k is encoded in binary, if necessary.

Safety. A game $\mathscr{G} = (S, s_0, S_c, S_e, \Sigma_c, \Sigma_e, \rightarrow, \mathrm{Win})$ with winning condition Win is a safety game if Win is accepted by a deterministic safety automaton.

Remark 8. *When Chen et al. introduced safety games under delayed control, they did not use automata to specify their winning plays, but instead equipped the game with a set of unsafe states and declared all those plays winning for controller that never visit an unsafe state. It is straightforward to see that our definition is equivalent, as their definition is captured by a deterministic safety automaton with two states. Conversely, taking the product of a game and a deterministic safety automaton yields an equivalent game with a state-based safety condition.*

Our results rely on the following two bounds on the transformations presented in Section 3, which are obtained by applying Remark 6:

1. If the winning condition Win for a game $\mathscr{G}$ under delayed control is given by a deterministic safety automaton with n states, then the winning condition $\overline{L(\mathscr{G})}$ is recognized by a deterministic reachability automaton with n states.
2. Dually, if the winning condition $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ of a delay game is given by a deterministic reachability automaton with n states, then the winning condition of the game $\mathscr{G}(\overline{L})$ under delayed action is recognized by a deterministic safety automaton with $\mathscr{O}(n \cdot |\Sigma_I|)$ states.

We begin by settling the complexity of determining whether controller wins a given safety game under every delay, which follows from the PSPACE-completeness of determining whether there is a lookahead that allows Player O to win a given delay game with reachability winning condition [9].

Theorem 1. *The following problem is* PSPACE*-complete: Given a safety game $\mathscr{G}$, does controller win $\mathscr{G}$ under every delay δ?*

Next, we give a lower bound on the complexity of determining whether controller wins a given safety game under a given delay, which is derived from a lower bound for delay games with reachability winning conditions.

Theorem 2. *The following problem is* PSPACE*-hard: Given a safety game $\mathscr{G}$ and δ (encoded in binary), does controller win $\mathscr{G}$ under delay δ.*

Note that we do not claim any upper bound on the problem considered in Theorem 2. There is a trivial 2EXPTIME upper bound obtained by hardcoding the delay into the graph of the safety game, thereby obtaining a classical delay-free safety game. It is open whether the complexity can be improved. Let us remark though that, via the correspondence to delay games presented in Section 3, improvements here would also yield improvements on the analogous problem for delay games, which is open too [15].

Next, we turn our attention to bounds on the delay for which controller wins. Recall that due to monotonicity, the set of delays for which controller wins is downward-closed, i.e., it is either a finite set $\{0, 2, 4 \dots, \delta_{\max}\}$ or it is equal to $2\mathbb{N}$. In the following, we present tight bounds on the value $\delta_{\max}$.

As a consequence, we settle a conjecture by Chen et al.: They conjectured that there is some delay δ_t (exponential in $|\mathscr{G}|$), such that if controller wins $\mathscr{G}$ under delay δ_t, then she wins under every delay. Note that this conjecture implies that $\delta_{\max}$ is at most exponential.

The following theorem proves Chen et al.'s conjecture, while Theorem 4 shows that δ_t must necessarily be exponential. For $\delta_{\max}$ this means it is at most exponential for every game, and can be exponential for some games.

The following two results are again obtained from similar bounds for delay games with reachability winning conditions.

Theorem 3. *Let $\mathscr{G}$ be a safety game. There is a $\delta_t \in \mathscr{O}(2^{|\mathscr{G}|})$ such that if controller wins $\mathscr{G}$ under delay δ_t, then she wins $\mathscr{G}$ under every δ.*

Finally, we show that the exponential upper bound on $\delta_{\max}$ is tight.

Theorem 4. *For every $n > 1$, there is a safety game $\mathscr{G}_n$ of size $\mathscr{O}(n)$ such that controller wins $\mathscr{G}$ under delay 2^n, but not under delay $2^n + 2$.*

Parity. Next, we consider the case of ω-regular winning conditions, given by deterministic parity automata. Applying Remark 6 yields the following two bounds on the transformations from Section 3:

1. If the winning condition Win for a game $\mathscr{G}$ under delayed control is given by a deterministic parity automaton with n states, then the winning condition $\overline{L(\mathscr{G})}$ is recognized by a deterministic parity automaton with n states.
2. Dually, if the winning condition $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ of a delay game is given by a deterministic parity automaton with n states, then the winning condition of the game $\mathscr{G}(\overline{L})$ under delayed action is recognized by a deterministic parity automaton with $\mathscr{O}(n \cdot |\Sigma_I|)$ states.

Exponential lookahead is both sufficient to win all ω-regular delay games that can be won and required to win some of these games [9]. Furthermore, determining whether there is some lookahead that allows Player O to win a given ω-regular delay game is ExpTime-complete [9]. As in the case of safety games, we can transfer these results to games under delayed control with ω-regular winning conditions.

Theorem 5.

1. *The following problem is* ExpTime*-complete: Given a game $\mathscr{G}$ with ω-regular winning condition specified by a deterministic parity automaton, does controller win $\mathscr{G}$ under every delay δ?*
2. *Let $\mathscr{G}$ be a game with ω-regular winning condition specified by a deterministic parity automaton with n states. There is a $\delta_t \in \mathscr{O}(2^{n^2})$ such that if controller wins $\mathscr{G}$ under delay δ_t, then she wins $\mathscr{G}$ under every δ.*
3. *For every $n > 1$, there is a game $\mathscr{G}_n$ of size $\mathscr{O}(n^2)$ with ω-regular winning condition specified by a two-state deterministic parity automaton $\mathscr{A}_n$ such that controller wins $\mathscr{G}$ under delay 2^n, but not under delay $2^n + 2$.*

Note that the lower bound on δ_t is just a restatement of Theorem 4, as safety games have ω-regular winning conditions.

Linear Temporal Logic. Finally, one can also transfer the triply-exponential upper and lower bounds on the necessary lookahead in delay games with LTL winning conditions as well as the 3EXPTIME-completeness of determining whether Player O wins such a delay game with respect to some lookahead [10] to games under delayed control with LTL winning conditions. Here, we exploit the following facts:

1. If the winning condition Win for a game $\mathscr{G}$ under delayed control is given by an LTL formula φ, then the winning condition $\overline{L(\mathscr{G})}$ is given by an LTL formula of size $\mathscr{O}(|\varphi|)$.
2. Dually, if the winning condition $L \subseteq (\Sigma_I \times \Sigma_O)^\omega$ of a delay game is given by an LTL formula φ, then the winning condition of the game $\mathscr{G}(\overline{L})$ under given action is given by an LTL formula of size $\mathscr{O}(|\varphi|)$.

Theorem 6.

1. *The following problem is* 3EXPTIME*-complete: Given a game $\mathscr{G}$ with winning condition specified by an LTL formula φ, does controller win $\mathscr{G}$ under every delay δ?*
2. *Let $\mathscr{G}$ be a game with ω-regular winning condition specified by an LTL formula φ. There is a $\delta_t \in \mathscr{O}(2^{2^{2^{|\varphi|+|\mathscr{G}|}}})$ such that if controller wins $\mathscr{G}$ under delay δ_t, then she wins $\mathscr{G}$ under every δ.*
3. *For every $n > 1$, there is a game $\mathscr{G}_n$ of size $\mathscr{O}(n^2)$ with winning condition specified by an LTL formula φ_n of size $\mathscr{O}(n^2)$ such that controller wins $\mathscr{G}$ under delay $2^{2^{2^n}}$, but not under delay $2^{2^{2^n}}+2$.*

To conclude, let us just remark that the results presented here also allow us to transfer results obtained for delay games with quantitative winning conditions [10, 13, 14] to games under delayed control with quantitative winning conditions. In fact, our result works for any winning condition, as long as the two transformations described in Section 3 are effective.

4.2 Environment's View

In Section 3, we proved a tight correspondence between controller in a game under delayed control and Player I in a delay game. Thus, it is natural to ask whether environment and Player O also share such a tight correspondence. A first indication that this is not the case can be obtained by considering the determinacy of these games: While delay games with Borel winning conditions are determined [8], even safety games under delayed action are not necessarily determined [2].

Upon closer inspection, this is not surprising, as the strategies in games under delayed control are not dual between the players: controller is at a disadvantage as she only gets delayed access to the actions picked by environment while environment does not benefit from this disadvantage. He does not get access to the actions picked by controller in advance. In a delay game however, the strategy definitions are completely dual: Player I has to grant lookahead on her moves which Player O gets access to. Thus, environment is in a weaker position than Player O.[1]

In this section, we study the correspondence between environment and Player O in detail by formally proving that environment is weaker than Player O.

Lemma 3. *Let $\mathscr{G}$ be a safety game. If environment wins $\mathscr{G}$ then Player O wins $\Gamma_k(\overline{L(\mathscr{G})})$ for every k.*

Now, we show that the converse direction fails.

[1]The difference can be formalized in terms of the information the players have access to: safety games under delay are incomplete-information games while delay games are complete-information games. Although interesting, we do not pursue this angle any further.

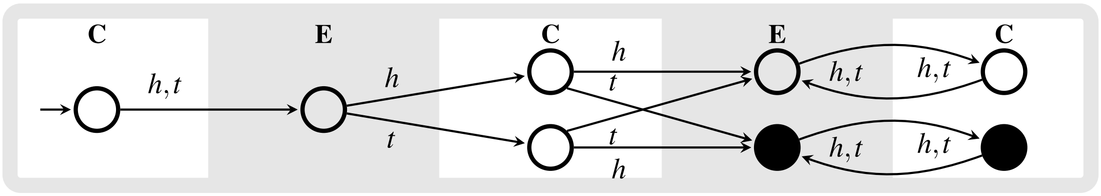


Figure 2: A safety game that environment does not win, but Player O wins the associated delay game. The initial state is marked by an arrow and the unsafe vertices are black. Note that both players have the actions h and t available.

Lemma 4. *There is a safety game $\mathscr{G}$ such that Player O wins $\Gamma_k(\overline{L(\mathscr{G})})$ for some k, but environment does not win $\mathscr{G}$.*

Proof. Let $\mathscr{G}$ be the safety game depicted in Fig. 2. With each move, the players place a coin (by either picking heads or tails) and environment wins a play by correctly predicting the second action of controller with his first action. Clearly, environment has no winning strategy in $\mathscr{G}$ because he has no access to future moves of controller. Stated differently, if environment picks h (t) in his first move, then the play in which the second action of controller is t (h) is winning for controller.[2]

Now, we consider the delay game $\Gamma_k(\overline{L(\mathscr{G})})$ for $k = 1$. Recall that the winning condition $\overline{L(\mathscr{G})}$ contains the winning plays for Player O, i.e., we have $\binom{\sigma_0\sigma_2\sigma_4\cdots}{\sigma_1\sigma_3\sigma_5\cdots} \in \overline{L(\mathscr{G})}$ if and only if $\sigma_1 \neq \sigma_2$. It is easy to see that Player O has a winning strategy in $\Gamma_k(\overline{L(\mathscr{G})})$ by simply flipping the second letter picked by Player I. This is possible since Player I has to provide two letters during the first round. □

Remark 9. *The safety game $\mathscr{G}$ depicted in Fig. 2 is in fact undetermined under every delay $\delta > 0$. In the proof of Lemma 4, we have already established that environment does not win $\mathscr{G}$. Now, under every delay $\delta > 0$, controller has to fix at least two actions before getting access to the first action picked by environment. This implies that there is, for every strategy for controller under delay δ, at least one consistent play that is losing for her, i.e., a play in which environment picks h (t) if the second move fixed by controller is t (h). Thus, no strategy is winning for controller under delay δ.*

Let us remark that, according to our definition of environment strategies, he is not able to enforce a losing play for controller (the game is undetermined after all), as he does not get access to the second action fixed by controller. Also, this is again the difference to delay games: Player O has access to these first two actions when making his first move, and is thereby able to win.

The full relation between games under delayed control and delay games is depicted in Fig. 3, restricted to Borel winning conditions (note that both transformations described in Section 3 preserve Borelness). The equivalence between controller winning the game under delayed control and Player I winning the corresponding delay game has been shown in Lemma 1 and Lemma 2. Also, Lemma 2 and Remark 7 imply that undetermined safety games under delayed control and those won by environment get transformed into delay games that are won by Player O. Finally, Lemma 1 and Remark 7 imply that delay games won by Player O get transformed into undetermined safety games under delayed control or to ones that are won by environment.

[2]Note that under any delay $\delta > 0$, controller cannot do this strategically, as she has to fix her first two actions in advance. But, as environment has no access to these fixed actions, he cannot react to them strategically.

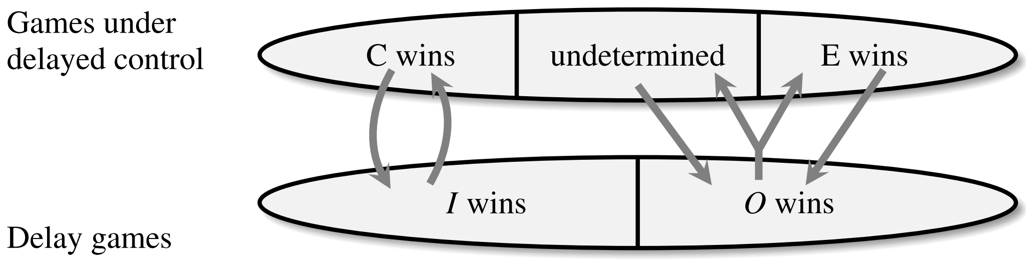


Figure 3: The relation between games under delayed control and delay games with Borel winning conditions. The upper ellipsis contains pairs $(\mathscr{G}, \delta)$ consisting of a game $\mathscr{G}$ under delayed control and a fixed delay δ; the lower one contains delay games $\Gamma_k(L)$ for some fixed k. The arrows represent the two transformations described in Section 3.

5 Refining the Correspondence: Sure Winning and Almost Sure Winning

It should be noted that the above transformations of games under delayed control into delay games and vice versa hinge on the fact that environment in the game under delayed control could, though lacking recent state information to do so strategically, by mere chance play the very same actions that the informed Player O in the delay game plays in his optimal adversarial strategy. That this constitutes a fundamental difference becomes apparent if we consider almost sure winning instead of sure winning. Almost sure winning calls for the existence of a mixed strategy that wins with probability 1, i.e., may fail on a set of plays with measure 0. This is different from sure winning in the sense of the definition of winning strategies for games under delayed control in Section 2.1, which calls for a strategy that never fails.

Remark 10. *We introduce mixed strategies for games under delayed control only, as delay games (with Borel winning conditions) are determined, which means that mixed strategies do not offer any advantage over pure strategies as introduced in Section 2.2.*

Given an even $\delta \geq 0$, a mixed strategy for controller in $\mathscr{G}$ under delay δ is a pair (α, τ_c) where $\alpha \in \mathscr{P}\left((\Sigma_c)^{\frac{\delta}{2}}\right)$ is a probability distribution over $(\Sigma_c)^{\frac{\delta}{2}}$ and $\tau_c\colon \mathrm{Pref}_c(\mathscr{G}) \to \mathscr{P}(\Sigma_c)$ maps play prefixes ending in S_c to probability distributions over actions of controller. A mixed strategy for environment is a mapping $\tau_e\colon \mathrm{Pref}_e(\mathscr{G}) \to \mathscr{P}(\Sigma_e)$.

The notion of consistency of a play with a strategy simply carries over, now inducing a Markov chain due to the probabilistic nature of the strategies. We say that a mixed strategy for controller (environment) *wins almost surely* if and only if it wins against any strategy of its opponent environment (controller) with probability 1, i.e., if and only if the winning condition is satisfied with probability 1 over the Markov chain induced by the game and the particular strategy combination. In this section, we write sure winning for winning as defined in Section 2, as is usual for games with randomized strategies.

The notion of almost sure winning alters chances for the players substantially by excluding the possibility of reliably playing an optimal strategy though lacking the information for doing so due to delayed observation. This can be seen from the following lemma, stating a fundamental difference between controller's power in games under delayed control and Player I's power in the corresponding delay games.

Lemma 5. *There is a game $\mathscr{G}$ under delayed control such that controller wins $\mathscr{G}$ almost surely under some delay δ while Player O (not Player I, which is the player corresponding to controller) wins the corresponding delay game $\Gamma_k(\overline{L(\mathscr{G})})$ for $k = \frac{\delta}{2}$, and surely so.*

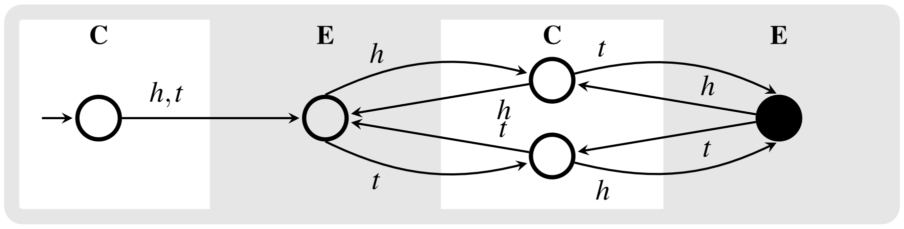


Figure 4: A reachability game that, under any positive delay, is won by controller almost surely via the simple randomized strategy of coin tossing (thus randomly generating head and tail events h and t), but won by player O surely if interpreted as a delay game due to the lookahead on Player I's actions granted to Player O. The initial state is marked by an arrow and controller wins if and only if the black vertex is visited at least once.

Proof. Consider the reachability game in Fig. 4 under delay 2 (or any larger delay). Intuitively, the players place a coin in each round (by picking either heads to tails with each move) and controller wins a play if the black state is visited, which happens if she selects a different coin placement than chosen by environment in the previous move.

Under any even (by definition) positive delay, controller wins this game with probability 1, i.e., almost surely, by a simple randomized strategy of coin tossing: by in each step randomly selecting action h or t with positive probability each, an eventual visit of the black state is guaranteed with probability 1, irrespective of being uninformed about environment's preceding move due to the delay.

The corresponding delay game $\Gamma_k(\overline{L(\mathscr{G})})$ for $k = \frac{\delta}{2}$, however, is easily won by Player O, because in delay games, the delayed Player I grants a lookahead to Player O. Hence, Player O can, due to the delay, already see the next move of Player I such that he can simply copy the next coin placement by Player I, safely staying in the non-black states and thereby win. □

Note that Lemma 5 implies that the previously observed correspondence between Player I and controller breaks down when considering almost sure winning strategies instead of just sure winning strategies: Games under delayed control for which Player O wins the corresponding delay game, are no longer either undetermined or won by environment, but may well be won by controller almost surely.

This consequently refines the correspondence between games under delayed control and delay games shown in Fig. 3 as follows.

Theorem 7. *Given a game $\mathscr{G}$ and an even $\delta \geq 2$, the following correspondences between $\mathscr{G}$ and the corresponding delay game $\Gamma_k(\overline{L(\mathscr{G})})$ for $k = \frac{\delta}{2}$ hold:*

1. *Controller surely wins $\mathscr{G}$ under delay δ if and only if Player I surely wins $\Gamma_k(\overline{L(\mathscr{G})})$.*
2. *If controller almost surely wins $\mathscr{G}$ under delay δ but cannot surely win $\mathscr{G}$ under delay δ then Player O surely wins $\Gamma_k(\overline{L(\mathscr{G})})$.*
3. *If environment surely or almost surely wins $\mathscr{G}$ under delay δ then Player O wins $\Gamma_k(\overline{L(\mathscr{G})})$.*
4. *If $\mathscr{G}$ is undetermined under delay δ with respect to almost sure winning strategies then Player O wins $\Gamma_k(\overline{L(\mathscr{G})})$.*
5. *All the aforementioned classes are non-empty, i.e., there exist games under delayed control where controller wins, where controller wins almost surely (but not surely), where environment wins surely, where environment wins almost surely (but not surely), and games which are undetermined with respect to almost-sure winning strategies.*

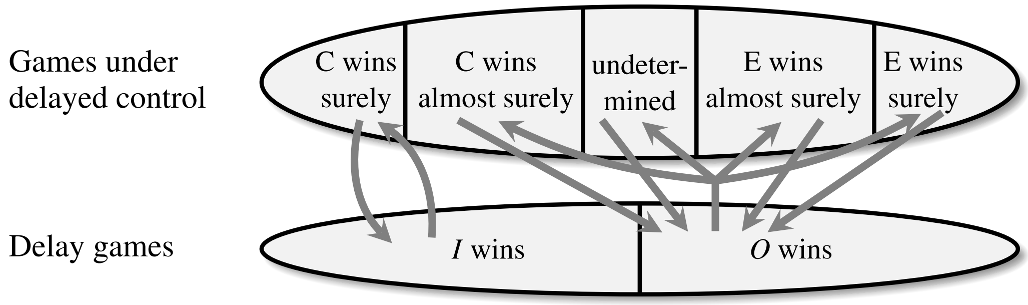


Figure 5: The relation between safety games under delayed control and delay games with Borel winning conditions. The upper ellipsis contains pairs $(\mathscr{G},\delta)$ consisting of a game $\mathscr{G}$ under delayed control and a fixed delay δ; the lower one contains delay games $\Gamma_k(L)$ for some fixed k. The arrows represent the two transformations described in Section 3.

The above correspondences are depicted in Fig. 5.

Item 2. of the above lemma is of particular interest, as it expresses a delay-related strengthening of controller relative to Player I, letting controller win almost surely where Player I looses for sure. The correspondence between controller and Player I observed in the deterministic setting thus breaks down when almost sure winning is considered and mixed strategies are permitted.

Remark 11. *In contrast to games under delayed control, where mixed strategies provide additional power to both the controller and the environment, the notions of sure winning and almost sure winning coincide for delay games (with Borel winning conditions) due to their determinacy [8]. Admitting mixed strategies (and almost sure winning) does not provide additional power to either of the two players in a delay game, as the determinacy result always implies existence of an infallible pure strategy for one of the players.*

6 Conclusion

We have compared delay games [9] and games under delayed control [2], two types of infinite games aiming to model asynchronicity in reactive synthesis, and have exhibited the differences in definitions and charted the relation between them with respect to both deterministic and randomized strategies: One can efficiently transform a game under delayed control into a delay game such that controller wins the game under delayed control with delay δ by a deterministic strategy if and only if Player I wins the resulting delay game with lookahead of size $\frac{\delta}{2}$. Dually, one can efficiently transform a delay game into a game under delayed control such that Player I wins the delay game with lookahead of size δ if and only if controller wins the resulting game under delayed control with delay 2δ by a deterministic strategy. These results allow us to transfer known complexity results and bounds on the amount of delay from delay games to games under delayed control, for which no such results were known, when considering deterministic strategies. We also proved that the analogous results fail in the setting of randomized strategies and almost sure winning conditions, as well as for the relation between environment and Player O, both under deterministic and randomized strategies.

Acknowledgements: Martin Fränzle has been supported by Deutsche Forschungsgemeinschaft under grant no. DFG FR 2715/5-1 “Konfliktresolution und kausale Inferenz mittels integrierter sozio-technischer Modellbildung”. Sarah Winter is a postdoctoral researcher at F.R.S.-FNRS. Martin Zimmermann has been supported by DIREC – Digital Research Centre Denmark.

References

[1] J. Richard Büchi & Lawrence H. Landweber (1969): *Solving Sequential Conditions by Finite-State Strategies*. Trans. Amer. Math. Soc. 138, pp. pp. 295–311, doi:10.2307/1994916.

[2] Mingshuai Chen, Martin Fränzle, Yangjia Li, Peter Nazier Mosaad & Naijun Zhan (2021): *Indecision and delays are the parents of failure - taming them algorithmically by synthesizing delay-resilient control*. Acta Informatica 58(5), pp. 497–528, doi:10.1007/s00236-020-00374-7.

[3] Martin Fränzle, Sarah Winter & Martin Zimmermann (2023): *Strategies Resilient to Delay: Games under Delayed Control vs. Delay Games*. arXiv 2305.19985, doi:10.48550/arXiv.2305.19985.

[4] David Gale & Frank M. Stewart (1953): *Infinite games with perfect information*. Annals of Mathematics 28, pp. 245–266, doi:10.1515/9781400881970-014.

[5] Erich Grädel, Wolfgang Thomas & Thomas Wilke, editors (2002): *Automata, Logics, and Infinite Games: A Guide to Current Research*. LNCS 2500, Springer, doi:10.1007/3-540-36387-4.

[6] Michael Holtmann, Lukasz Kaiser & Wolfgang Thomas (2012): *Degrees of Lookahead in Regular Infinite Games*. Log. Methods Comput. Sci. 8(3), doi:10.2168/LMCS-8(3:24)2012.

[7] Frederick A. Hosch & Lawrence H. Landweber (1972): *Finite Delay Solutions for Sequential Conditions*. In Maurice Nivat, editor: ICALP 1972, North-Holland, Amsterdam, pp. 45–60.

[8] Felix Klein & Martin Zimmermann (2015): *What are Strategies in Delay Games? Borel Determinacy for Games with Lookahead*. In Stephan Kreutzer, editor: CSL 2015, LIPIcs 41, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 519–533, doi:10.4230/LIPIcs.CSL.2015.519.

[9] Felix Klein & Martin Zimmermann (2016): *How Much Lookahead is Needed to Win Infinite Games?* Log. Methods Comput. Sci. 12(3), doi:10.2168/LMCS-12(3:4)2016.

[10] Felix Klein & Martin Zimmermann (2016): *Prompt Delay*. In Akash Lal, S. Akshay, Saket Saurabh & Sandeep Sen, editors: FSTTCS 2016, LIPIcs 65, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 43:1–43:14, doi:10.4230/LIPIcs.FSTTCS.2016.43.

[11] Robert McNaughton (2000): *Playing Infinite Games in Finite Time*. In Arto Salomaa, Derick Wood & Sheng Yu, editors: A Half-Century of Automata Theory: Celebration and Inspiration, World Scientific, pp. 73–91.

[12] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: FOCS 1977, IEEE Computer Society, pp. 46–57, doi:10.1109/SFCS.1977.32.

[13] Martin Zimmermann (2016): *Delay Games with WMSO+U Winning Conditions*. RAIRO Theor. Informatics Appl. 50(2), pp. 145–165, doi:10.1051/ita/2016018.

[14] Martin Zimmermann (2017): *Games with costs and delays*. In: LICS 2017, IEEE Computer Society, pp. 1–12, doi:10.1109/LICS.2017.8005125.

[15] Martin Zimmermann (2022): *Approximating the minimal lookahead needed to win infinite games*. Information Processing Letters 177, p. 106264, doi:10.1016/j.ipl.2022.106264.

Fast Algorithms for Energy Games in Special Cases[*]

Sebastian Forster
Department of Computer Science
University of Salzburg, Austria
sebastian.forster@plus.ac.at

Antonis Skarlatos
Department of Computer Science
University of Salzburg, Austria
antonis.skarlatos@plus.ac.at

Tijn de Vos
Department of Computer Science
University of Salzburg, Austria
tijn.devos@plus.ac.at

In this paper, we study algorithms for special cases of energy games, a class of turn-based games on graphs that show up in the quantitative analysis of reactive systems. In an energy game, the vertices of a weighted directed graph belong either to Alice or to Bob. A token is moved to a next vertex by the player controlling its current location, and its energy is changed by the weight of the edge. Given a fixed starting vertex and initial energy, Alice wins the game if the energy of the token remains nonnegative at every moment. If the energy goes below zero at some point, then Bob wins. The problem of determining the winner in an energy game lies in $\mathsf{NP} \cap \mathsf{coNP}$. It is a long standing open problem whether a polynomial time algorithm for this problem exists.

We devise new algorithms for three special cases of the problem. The first two results focus on the single-player version, where either Alice or Bob controls the whole game graph. We develop an $\tilde{O}(n^{\omega} W^{\omega})$ time algorithm for a game graph controlled by Alice, by providing a reduction to the All-Pairs Nonnegative Prefix Paths problem (APNP), where W is the maximum absolute value of any edge weight and ω is the best exponent for matrix multiplication. Thus we study the APNP problem separately, for which we develop an $\tilde{O}(n^{\omega} W^{\omega})$ time algorithm. For both problems, we improve over the state of the art of $\tilde{O}(mn)$ for small W. For the APNP problem, we also provide a conditional lower bound which states that there is no $O(n^{3-\varepsilon})$ time algorithm for any $\varepsilon > 0$, unless the APSP Hypothesis fails. For a game graph controlled by Bob, we obtain a near-linear time algorithm. Regarding our third result, we present a variant of the value iteration algorithm, and we prove that it gives an $O(mn)$ time algorithm for game graphs without negative cycles, which improves a previous upper bound. The all-Bob algorithm is randomized, all other algorithms are deterministic.

1 Introduction

Energy games belong to a class of turn-based games on graphs that show up in the quantitative analysis of reactive systems. A game graph can possibly represent a scheduling problem, where vertices are the configurations of the system and edges carry positive or negative values representing the evolution of resources. Thus, in this model resources can be consumed or produced. The energy games problem has been introduced in the early 2000s [13, 6], but also have been implicitly studied before due to its ties to mean-payoff games [21]. Energy games have applications in, among others, computer aided verification and automata theory [13, 5, 12], and in online and streaming problems [31]. From its computational perspective, the problem of determining the winner in an energy game lies in $\mathsf{NP} \cap \mathsf{coNP}$. It is an intriguing open problem whether a polynomial time algorithm for this problem exists.

An energy game is played by two players, say Alice and Bob, on a *game graph*, which is a weighted directed graph such that each vertex is either controlled by Alice or Bob. The game starts by placing a token with an initial energy on a starting vertex. The game is played in rounds, and every time the token

[*]This work is supported by the Austrian Science Fund (FWF): P 32863-N. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 947702). A full version is available at `https://arxiv.org/abs/2307.08442`.

is located at a vertex controlled by Alice, then Alice chooses the next location of the token among the outgoing edges, otherwise Bob chooses the next move. The token has an *energy level* (in the beginning this is equal to the initial energy) and every time it traverses an edge, the weight of the edge is added to the energy level (a negative weight amounts to a reduction of the energy level). The objectives of the players are as follows: Alice wants to minimize the initial energy that is necessary to keep the energy level nonnegative at all times, whereas Bob wants to maximize this value (and possibly drive it to ∞). The computational problem is to determine for each vertex the minimum initial energy such that Alice can guarantee against all choices of Bob that the energy level always stays nonnegative.

Energy games are a generalization of parity games [24, 9], polynomial-time equivalent to mean-payoff games [6, 9], and a special case of simple stochastic games [31]. Recent progress on parity games yielded several quasipolynomial time algorithms [11], but the corresponding techniques seem to not carry over to energy and mean-payoff games [20]. Consequently, the complexity of energy games is still "stuck" at pseudopolynomial [9] or subexponential time [4]. Hence, in this paper we focus on interesting special cases (which are non-trivial problems) that admit fast algorithms. Two of these cases are game graphs where all vertices are controlled by one player, and the third case are game graphs with no negative cycles.

All-Pairs Nonnegative Prefix Paths. We also study another reachability problem with energy constraints [22, 17], the *All-Pairs Nonnegative Prefix Paths (APNP) problem*. In this problem, the goal is to find for every pair of vertices whether there exists a path π such that the weight of each prefix of π is nonnegative. We use this problem to obtain the result for the special case where Alice controls the whole game graph, since the two problems are closely related. Dorfman, Kaplan, Tarjan, and Zwick [17] solve the more general problem, where for each pair of vertices the goal is to find the path π of maximum weight among all options. This problem naturally generalizes APSP, and they solve it in $O(mn+n^2\log n)$ time.

Energy Games. The state-of-the-art algorithms for the energy games are either deterministic with running time $O\left(\min(mnW, mn2^{n/2}\log W)\right)$ [9, 18] or randomized with subexponential running time $2^{O(\sqrt{n\log n})}$ [4]. Special cases of the energy games have been studied by Chatterjee, Henzinger, Krinninger, and Nanongkai [15]. They present a variant of the value iteration algorithm of [9] with running time $O(m|A|)$, where A is a sorted list containing all possible minimum energy values. This does not improve the general case, as A in the worst case is the list $\{0,1,\ldots,nW,\infty\}$. However, it does give a faster running time if the weights adhere to certain restrictions. Moreover, they develop a scaling algorithm with running time $O(mn\log W(\log\frac{W}{P}+1)+mn\frac{W}{P})$, where $P\in\{\frac{1}{n},\ldots,W\}$ is a lower bound on the *penalty* of the game.

For the special case where there are no negative cycles in the game graph, the penalty can be set to W, and the scaling algorithms of [15] solves the problem in $O(mn\log W)$ time. For another special case where the whole game graph is controlled by Alice, Brim and Chaloupka [8] provided an $\tilde{O}(mn)$[1] running time algorithm as a subroutine for the two-players version.

Mean-Payoff Games. In a mean-payoff game, the objective of Alice is to maximize the average of the weights of edges traversed so far, whereas Bob's objective is to minimize this mean payoff. It is well known that any energy games instance can be solved by solving $O(n\log(nW))$ mean-payoff games [6],

[1]We write $\tilde{O}(f)$ for $O(f\,\mathrm{poly}\log f)$.

and any mean-payoff game instance can be solved by solving $O(\log(nW))$ energy games with maximal weight nW [9][2]. Thus, any of the aforementioned algorithms for solving energy games also yields an algorithm for solving mean-payoff games at the expense of at most an additional factor of $O(n\log(nW))$ in the running time. Zwick and Paterson [31] provided the first pseudopolynomial time algorithm that computes all the mean-payoff values, with $O(mn^3W)$ running time. Later, the running time was improved by Brim, Chaloupka, Doyen, Gentiline, and Raskin [9] to $O(mn^2W\log(nW))$, using their reduction to energy games. The state-of-the-art algorithm for solving a mean-payoff game is due to Comin and Rizzi [16] which runs in $O(mn^2W)$ time.

1.1 Our Results and Techniques

All-Pairs Nonnegative Prefix Paths. The version of All-Pairs Nonnegative Prefix Paths (APNP) problem where we want to find the path of maximum weight [17], naturally generalizes the All-Pairs Shortest Paths (APSP) problem. The APSP Hypothesis states that there is no $O(n^{3-\varepsilon})$ time algorithm for the APSP, for any $\varepsilon > 0$. However, this version of APNP is more than what is necessary for the application of energy games. We show that the weaker version which only computes reachability (as APNP has been defined), also does not allow for a $O(n^{3-\varepsilon})$ time algorithm for any $\varepsilon > 0$, under the APSP Hypothesis.

Theorem 1.1. *Unless the APSP Hypothesis fails, there is no $O(n^{3-\varepsilon})$ time algorithm that solves the All-Pairs Nonnegative Prefix Paths problem, for any $\varepsilon > 0$.*

We parameterize the maximum absolute value of any edge weight W, and we obtain an algorithm with a faster running time for small values of W.

Theorem 1.2. *There exists a deterministic algorithm that, given a graph $G = (V,E,w)$ with edge weights in the interval $[-W,W]$, solves the All-Pairs Nonnegative Prefix Paths problem in $\tilde{O}(n^{\omega}W^{\omega})$ time.*

All-Alice. Our first contribution regarding the special cases of energy games, concerns the *all-Alice case* in which all vertices are controlled by Alice. Note that if we fix a strategy for Bob in any game graph, this can be seen as an all-Alice instance.

Theorem 1.3. *There exists a deterministic algorithm that, given a game graph $G = (V,E,w)$ in which all vertices are controlled by Alice, computes the minimum sufficient energy of all vertices in $\tilde{O}(n^{\omega}W^{\omega})$ time.*

Note that the aforementioned reduction from energy games to mean-payoff games always introduces Bob vertices. Thus, algorithms for the all-Alice mean-payoff decision problem cannot be leveraged by this reduction to compute the minimum energies in the all-Alice case.

Our approach for the all-Alice case consists of two steps. In the first step, we identify all vertices Z such that minimum initial energy 0 suffices, by using Theorem 1.2. In the second step, we compute the paths of least energy reaching any vertex in Z. For small values of W, this improves on the state-of-the-art $\tilde{O}(mn)$ algorithm [8].

All-Bob. Our second contribution regarding the special cases of energy games, is a faster algorithm for the *all-Bob case* in which all vertices are controlled by Bob. Note that if we fix a strategy for Alice in any game graph, this can be seen as an all-Bob instance.

[2]Unless stated otherwise, we always consider the versions of the games where we compute the mean-payoff value/minimum initial energy for *all* vertices.

Theorem 1.4. *There exists a randomized (Las Vegas) algorithm that, given a game graph $G = (V, E, w)$ in which all vertices are controlled by Bob, computes the minimum sufficient energy of all vertices, and with high probability the algorithm takes $O(m \log^2 n \log nW \log\log n)$ time.*

To the best of our knowledge, the fastest known algorithm for the all-Bob case is implied by the reduction to the mean-payoff decision problem and has a running time of $\tilde{O}(mn \log^2 W)$. This comes from $\tilde{O}(n \log W)$ calls to the state-of-the-art negative-cycle detection algorithm [3, 10].

Our approach for the all-Bob case consists of two steps. In the first step, we run a negative-cycle detection algorithm to remove all vertices reaching a negative cycle. In the second step, we add an artificial sink to the graph with an edge from every vertex to the sink, and we compute the shortest path of every vertex to the sink using a single-source shortest paths (SSSP) algorithm. Note that this construction is very close to Johnson's method for computing suitable vertex potentials [23]. Further note that, since energy games are not symmetric for Alice and Bob, our near-linear time all-Bob algorithm has no implications for the all-Alice case.

No Negative Cycles. Finally, we give an improved algorithm for the special case where there are no negative cycles.

Theorem 1.5. *There exists a deterministic algorithm that, given a grame graph $G = (V, E)$ without negative cycles, computes the minimum sufficient energy of all vertices in $O(mn)$ time.*

To the best of our knowledge, the fastest known algorithm for this special case has a running time of $O(mn \log W)$ by running the above mentioned algorithm of Chatterjee, Henzinger, Krinninger, and Nanongkai [15] with penalty $P = W$. We use a new variant of the value iteration algorithm where the energy function after i steps corresponds to the minimum energy function in an *i-round game*. A similar variant has been used by Chatterjee, Doyen, Randour, and Raskin [14] for the Mean-Payoff games. We adapt this algorithm and provide the necessary analysis to use it for energy games.

An i-round game is a finite version of the energy game, where a token is passed for only i rounds. In this version, the goal is to find the initial energy that Alice needs, in order to keep the energy level nonnegative for these i-rounds. Then we show that in graphs without negative cycle, the infinite game is equivalent to the n-round game.

Structure of the paper. In the next section, we provide some preliminaries, including the formal definition of an energy game. In Section 3, we study the All-Pairs Nonnegative Prefix Paths problem, and we present an algorithm for the special case that the edge weights are in $\{-1, 0, +1\}$, an algorithm for general edge weights, and a lower bound. Next in Section 4, we consider the all-Alice case by reducing this problem to the All-Pairs Nonnegative Prefix Paths problem. In Section 5, we consider the all-Bob case, and finally in Section 6, we consider game graphs without negative cycles.

2 Preliminaries

Graphs. Given a directed graph $G = (V, E, w)$, we denote by $n = |V|$ the number of vertices, by $m = |E|$ the number of edges, and by W the maximum absolute value of any edge weight. Also, we denote $N^+(v)$ for the *out-neighborhood* of v, i.e., $N^+(v) := \{u \in V : (v, u) \in E\}$. Further, we denote $\deg^+(v)$ for the *out-degree* of v, i.e., $\deg^+(v) := |N^+(v)|$. Similarly, $N^-(v)$ and $\deg^-(v)$ denote the *in-neighborhood* and *in-degree* respectively.

A *path* P is a sequence of vertices $u_0u_1\cdots$ such that $(u_i,u_{i+1})\in E$ for every $i\geq 0$. We say a path is *finite* if it contains a finite number of vertices (counted with multiplicity). We say a path is *simple* if each vertex appears at most once. A *lasso* is a path of the form $u_0u_1\cdots u_ju_i$, where the vertices $u_0,\ldots,u_j$ are disjoint and $i<j$. In other words, it is a simple path leading to a cycle. A *nonnegative prefix path* is a path $P=u_0u_1\cdots$ such that $\sum_{j=0}^{i-1}w(u_j,u_{j+1})\geq 0$ for all $1\leq i\leq |P|$. Further, we denote the weight of a path $P=u_0u_1\cdots$ by $w(P):=\sum_{j=0}^{|P|-1}w(u_j,u_{j+1})$. For a fixed path $P=u_0u_1\cdots$, the energy level $e(u_i)$ of a vertex u_i in P is equal to $\sum_{j=0}^{i-1}w(u_j,u_{j+1})$. That is, the sum of all the weights along P until u_i.

Let $G=(V,E,w)$ be a directed graph with edge weights -1 and $+1$, and let $s,t\in V$ be two vertices of G. Then, a *Dyck path from s to t* is a nonnegative prefix path from s to t of total weight zero [7]. For a graph H, we refer to the corresponding functions by using H as subscript (e.g., we use the notation $w_H(\cdot)$ for the weight function of H).

Energy Games. An energy game is an infinite duration game played by two players, Alice and Bob. The game is played on a *game graph* which a weighted directed graph $G=(V,E,w)$, where each vertex has at least one outgoing edge. The weights are integers and lie in the range $\{-W,-W+1,\ldots,W-1,W\}$. The set of vertices is partitioned in two sets V_A and V_B, controlled by Alice and Bob respectively. Furthermore, we are given a starting vertex $s\in V$, and initial energy $e_0\geq 0$. We start with position $v_0=s$. After the i_{th} round, we are at a position $v_i\in V$ and have energy e_i. In the i_{th} round, if $v_{i-1}\in V_A$ ($v_{i-1}\in V_B$) then Alice (Bob) chooses a next vertex $v_i\in N^+(v_{i-1})$ and the energy changes to $e_i=e_{i-1}+w(v_{i-1},v_i)$. The game ends when $e_i<0$, in which case we say that Bob wins. If the game never ends, namely, $e_i\geq 0$ for all $i\geq 0$, we say that Alice wins. The goal is to find out the minimum initial energy $e_0\geq 0$ such that Alice wins when both players play optimally. Note that allowing $e_0=\infty$ means that such an energy always exist.

To make this goal more formal, we have to introduce *strategies*. A strategy for Alice (Bob) tells us given the current point $v_i\in V_A$ ($v_i\in V_B$) and the history of the game, $v_0,\ldots,v_i$, where to move next. It turns out that we can restrict ourselves to *positional strategies* [19, 6], which are deterministic and do not depend on the history of the game. We denote a positional strategy of Alice by $\sigma\colon V_A\to V$ where $\sigma(v)\in N^+(v)$ for $v\in V_A$, and a positional strategy of Bob by $\tau\colon V_B\to V$ where $\tau(v)\in N^+(v)$ for $v\in V_B$. For any pair of strategies (σ,τ) we define $G(\sigma,\tau)$ to be the subgraph (V,E') corresponding to these strategies, where $E'=\{(v,\sigma(v)):v\in V_A\}\cup\{(v,\tau(v)):v\in V_B\}$. Note that in this graph each vertex has exactly one out-neighbor. Let P_i be the unique path $s=u_0,u_1,\ldots,u_i$ in $G(\sigma,\tau)$ of length i originating at s. Then at vertex s with initial energy e_0 and with these strategies, Alice wins if $e_0+w(P_i)\geq 0$ for all $i\geq 0$, and Bob wins if $e_0+w(P_i)<0$ for at least one $i\geq 0$. The *minimum sufficient energy at s with respect to σ and τ* is the minimum energy such that Alice wins, namely $e_{G(\sigma,\tau)}(s):=\max\{0,-\inf_{i\geq 0}w(P_i)\}$. Finally, we define the *minimum sufficient energy* at s as follows:

$$e^*_G(s):=\min_{\sigma}\max_{\tau}e_{G(\sigma,\tau)}(s),$$

where the minimization and the maximization are over all the positional strategies σ of Alice and τ of Bob, respectively. We omit the subscript G, and use $e_{\sigma,\tau}(s)$ instead of $e_{G(\sigma,\tau)}(s)$, whenever this is clear from the context. By Martin's determinacy theorem [26], we have that $\min_\sigma\max_\tau e_{\sigma,\tau}(s)=\max_\tau\min_\sigma e_{\sigma,\tau}(s)$, thus the outcome is independent of the order in which the players pick their strategy. Now we can define *optimal strategies* as follows. A strategy σ^* is an optimal strategy for Alice, if $e_{\sigma^*,\tau}(s)\leq e^*(s)$ for any strategy τ of Bob. Similarly, τ^* is an optimal strategy for Bob, if $e_{\sigma,\tau^*}(s)\geq e^*(s)$ for any strategy σ of Alice. An *energy function* is a function $e\colon V\to\mathbb{Z}_{\geq 0}\cup\{\infty\}$. The function $e^*_G(\cdot)$ (or $e^*(\cdot)$) as defined above, is the *minimum sufficient energy function*.

3 All-Pairs Nonnegative Prefix Paths Problem

In this section, we study the All-Pairs Nonnegative Prefix Paths (APNP) problem. The goal of this problem is to find for every pair of vertices whether there exists a nonnegative prefix path between them. A similar problem is the *All-Pairs Dyck-Reachability problem*, where the goal is to find for every pair of vertices whether there exists a Dyck path between them (given that the edge weights are in $\{-1,+1\}$). Furthermore, another standard problem is the *transitive closure problem*, which asks to find for every pair of vertices whether there exists a path between them.

Bradford [7] provided an $\tilde{O}(n^{\omega})$ time algorithm for the All-Pairs Dyck-Reachability problem. Moreover, the transitive closure problem admits an $\tilde{O}(n^{\omega})$ algorithm [1].

Theorem 3.1. *There exists a deterministic algorithm that, given a graph $G=(V,E,w)$ with edge weights in $\{-1,1\}$, solves the All-Pairs Dyck-Reachability problem in $\tilde{O}(n^{\omega})$ time.*

Our approach for the APNP problem consists of two stages. At first, we solve the APNP problem for the special case where the edge weights are from the set $\{-1,0,+1\}$, by exploiting the algorithm of [7] for the All-Pairs Dyck-Reachability problem. Afterwards, we extend our algorithm to work with general weights, by showing that a reduction used in [2] preserves the properties we need.

In the end of the section, we also present a conditional lower bound for the APNP problem under the APSP Hypothesis, which is one of the main hypotheses in fine-grained complexity.

3.1 All-Pairs Nonnegative Prefix Paths with edge weights in $\{-1,0,+1\}$

Consider a graph $G=(V,E)$ with edge weights -1 and $+1$. By definition, we have that any Dyck path is also a nonnegative prefix path. However, the opposite is not necessarily true. Recall that nonnegative prefix paths allow the energy level of their last vertex to be a strictly positive value, while in Dyck paths this value must be zero. This implies that an All-Pairs Dyck-Reachability algorithm does not trivially gives us an All-Pairs Nonnegative Prefix Paths algorithm. Nevertheless, we show how to overcome this issue and we use an All-Pairs Dyck-Reachability algorithm as a subroutine in order to solve the All-Pairs Nonnegative Prefix Paths problem.

Algorithm for the $\{-1,0,+1\}$ case. Consider a directed graph $G=(V,E,w)$, with edge weights in $\{-1,0,+1\}$. In the beginning of the algorithm, we construct a graph G_2 as follows.

1. Initially, we create a new graph $G_1=(V_1,E_1,w)$ by replacing every edge of zero weight with an edge of weight $+1$ and an edge of weight -1. Specifically, for each vertex u with at least one outgoing edge $(u,v)\in E$ with $w(u,v)=0$, we add a new vertex u', and add an edge (u,u') with $w(u,u')=+1$. Next, for each edge $(u,v)\in E$ with $w(u,v)=0$, we remove the edge (u,v), and add the edge (u',v) with weight -1.[3]
2. Next, we run on G_1 the algorithm of Theorem 3.1, which solves the All-Pairs Dyck-Reachability in time $\tilde{O}(n^{\omega})$ for edge weights in $\{-1,+1\}$.
3. Finally, we create another new graph $G_2=(V,E_2)$ with the original vertex set and an edge set E_2 defined as follows. The set E_2 contains an edge $(u,v)\in V\times V$ if and only if there is a Dyck path from u to v in G_1 or $w(u,v)=1$ in G, if $(u,v)\in E$.

[3]Note that by doing the naive thing which is to replace each edge of zero weight by two edges, one with weight $+1$ and one with weight -1, potentially blows up the number of vertices to $\Omega(m)$. In turn, since the running time depends on the number of vertices, this translates to a blow up of the running time.

In the end, we run on G_2 a transitive closure algorithm, and we return that there is a nonnegative prefix path in G if and only if there is a path in G_2. Notice that graphs G and G_1 are weighted, while G_2 is unweighted.

Analysis of the algorithm. The following observation shows that the replacement of zero weight edges is valid, in the sense that nonnegative prefix paths of total weight zero[4] in G correspond to Dyck paths in G_1 and vice versa. Moreover, we prove the claim that the transitive closure problem in G_2 is equivalent to the All-Pairs Nonnegative Prefix Paths problem in G.

Observation 3.2. *For every pair of vertices $u, v \in V$, there exists a nonnegative prefix path of total weight zero from u to v in G if and only if there exists a Dyck path from u to v in G_1.*

Lemma 3.3. *For every pair of vertices $u, v \in V$, there exists a nonnegative prefix path from u to v in G if and only if there exists a path from u to v in G_2.*

Proof. Assume that there exists a nonnegative prefix path π from u to v in G. Let a be the first vertex after u along π with a minimum energy level. Initially, we show that the edge (u,a) appears in G_2. Since π is a nonnegative prefix path, we have that $e(u) \leq e(a)$. If $e(u) < e(a)$, then there must be an edge (u,a) in G with weight $+1$. Also if $e(u) = e(a)$, then the subpath of π from u to a is a nonnegative prefix path of total weight zero. Then by Observation 3.2, the subpath of π from u to a is a Dyck path in G_1. Therefore, in both cases we have added the edge (u,a) in G_2. As the vertex a has a minimum energy level, we can apply the same argument iteratively starting from a, to conclude that there exists a path from u to v in G_2.

Assume now that there exists a path π from u to v in G_2. By construction, the edges of π correspond either to edges in G with weight $+1$ or to Dyck paths in G_1. By Observation 3.2, these Dyck paths in G_1 correspond to nonnegative prefix paths of total weight zero in G. Since positive edges increase the energy level and nonnegative prefix paths at least maintain the energy level, we conclude that there exists a nonnegative prefix path from u to v in G. $\square$

Lemma 3.4. *There exists a deterministic algorithm that, given a graph $G = (V,E,w)$ with edge weights in $\{-1,0,+1\}$, solves the All-Pairs Nonnegative Prefix Paths problem in $\tilde{O}(n^{\omega})$ time.*

Proof. The number of vertices of G_1 is $O(n)$ by construction, where n is the initial number of vertices in G. Hence, the construction of G_2 runs in $\tilde{O}(n^{\omega})$. Moreover, the transitive closure problem in G_2 can be solved in $\tilde{O}(n^{\omega})$ time as well [1]. Thus by Lemma 3.3, the claim follows. $\square$

3.2 All-Pairs Nonnegative Prefix Paths with general edge weights

We extend now Lemma 3.4 for graphs with general edge weights, in the cost of an extra factor W^{ω} in the running time. The idea is to use the reduction by Alon, Galil, and Margalit [2], who reduce the All-Pairs Shortest Paths (APSP) problem with general edge weights to the special case where the edge weights are in $\{-1,0,+1\}$. We present the reduction for completeness, and we prove that the same reduction also preserves the properties that we need for the All-Pairs Nonnegative Prefix Paths problem.

[4]Observe that a Dyck path is a nonnegative prefix path of total weight zero consisting only of edges -1 and $+1$. Thus, we avoid to use the term *Dyck path* for G because it may contains edges of weight zero.

Reduction from general weights to $\{-1,0,+1\}$ [2]. Given a graph $G=(V,E,w)$ with weights in the interval $[-W,W]$, we create another graph G' with weights only in $\{-1,0,+1\}$, as follows. For every vertex $v \in V$ in G, we create $2W+1$ vertices $\{v^i\}_{i=-W}^{W}$ in G'. We say that vertex v^0 of G' is the origin of vertex v. Then, we add in G' an edge (v^{i+1},v^i) of weight -1, for every $-W \leq i \leq -1$, and an edge (v^{i-1},v^i) of weight 1, for every $1 \leq i \leq W$. Moreover, for every edge (u,v) of weight k in G, we add an edge (u^k,v^0) of zero weight in G'.

Theorem 1.2. *There exists a deterministic algorithm that, given a graph $G=(V,E,w)$ with edge weights in the interval $[-W,W]$, solves the All-Pairs Nonnegative Prefix Paths problem in $\tilde{O}(n^{\omega}W^{\omega})$ time.*

Proof. The idea is to apply the reduction mentioned above and use the algorithm of Lemma 3.4 in G'. Then, we claim that there exists a nonnegative prefix path from u to v in G if and only if there exists a nonnegative prefix path from u^0 to v^0 in G'.

Regarding the running time, since the number of vertices of the new graph G' after the reduction becomes $\Theta(nW)$, the running time of the algorithm becomes $\tilde{O}((nW)^{\omega})$. It remains to prove the correctness of the algorithm.

Let π be a nonnegative prefix path from u to v in G. We construct a path π' from u^0 to v^0 in G' as follows. For every edge $(a,b) \in \pi$ of weight k, we add to π' the unique subpath from a^0 to b^0 of weight k in G', which exists by construction. Since π is a nonnegative prefix path in G, and every subpath we add to π' consists either only of edges with weight in $\{-1,0\}$ or only of edges with weight in $\{0,+1\}$, we can infer that π' is a nonnegative prefix path from u^0 to v^0 in G'.

For the other direction, let π' be a nonnegative prefix path from u^0 to v^0 in G'. We construct a path π from u to v in G as follows. Let a^0 be the first vertex after u^0 along π' such that, a^0 is the origin vertex of a different vertex than u (i.e., a^0 is the origin of a vertex $a \neq u$). By construction, there exists an edge (u,a) of weight k in G, where k is the weight of the subpath from u^0 to a^0 in π'. We add the edge (u,a) in π, and continue with the construction of π by applying the same argument iteratively starting from a^0 until we reach v^0. Since π' is a nonnegative prefix path in G', and each prefix of π corresponds to a prefix in π', we can infer that π is a nonnegative prefix path from u to v in G.

Therefore, the pair of vertices $\{u^0,v^0\}$ in G' contains the information for the pair of vertices $\{u,v\}$ in G, and so the claim follows. ☐

3.3 Lower bound for All-Pairs Nonnegative Prefix Paths

We prove a lower bound on the running time of All-Pairs Nonnegative Prefix Paths problem under the APSP Hypothesis. The APSP Hypothesis is an assertions that the All-Pairs Shortest Paths (APSP) problem cannot be solved in truly subcubic $O(n^{3-\varepsilon})$ time, for any $\varepsilon > 0$. Vassilevska Williams and Williams [29] proved that APSP and Negative Triangle are equivalent under subcubic reductions. The Negative Triangle problem is defined as follows. Given a graph $G=(V,E,w)$, the goal is to find three vertices a,b,c such that $w(a,b)+w(b,c)+w(c,a) < 0$, that is, the vertices a,b,c form a negative weight cycle.

Recently, a reduction from the Negative Triangle problem to the h-hop-bounded s-t path problem was given by Polak and Kociumaka [25], in order to prove a hardness result for the latter. Motivated by this reduction, we also reduce the Negative Triangle problem to the All-Pairs Nonnegative Prefix Paths problem to obtain a hardness result for the All-Pairs Nonnegative Prefix Paths problem, as shown in Theorem 1.1.

We first provide an auxiliary lemma, which we also use later in Lemma 4.1.

Lemma 3.5. *Given a graph $G = (V, E, w)$, let C be a nonnegative weight cycle in G (i.e., $w(C) \geq 0$). Then, there is a vertex $u \in C$ in the cycle, such that there exists a nonnegative prefix path in G from u to itself along C.*

Proof. Let $Q \subseteq C$ be a subpath of C with the most negative total weight, and Q' be the rest of C (i.e., $Q \cup Q' = C$). Notice that the weight of all prefixes in Q' must be nonnegative, otherwise this negative weight prefix could be merged with Q, contradicting the fact that Q is the subpath of C with the most negative total weight. Moreover, as $w(C) \geq 0$ we have that $w(Q') \geq -w(Q)$. Since by definition of Q, there is no prefix of Q with more negative total weight, it holds that $Q' \cup Q$ is a nonnegative prefix path from the first vertex of Q' to itself along C. □

Theorem 1.1. *Unless the APSP Hypothesis fails, there is no $O(n^{3-\varepsilon})$ time algorithm that solves the All-Pairs Nonnegative Prefix Paths problem, for any $\varepsilon > 0$.*

Proof. Consider a Negative Triangle instance $G = (V, E)$. We create a directed graph $G_1 = (V_1, E_1)$ as follows. The vertex set V_1 of G_1 consists of five copies of all vertices, i.e., $V_1 := \{v^i : v \in V, i \in \{1,2,3,4,5\}\}$. For every edge $(u,v) \in E$ of weight $w(u,v)$, we add an edge (u^i, v^{i+1}) to E_1 with weight $-w(u,v)$, for $1 \leq i < 4$. Also for each vertex $v \in V$, we add an edge (v^4, v^5) of weight $w_{\min} = -1$.

We claim that there exists a negative weight triangle in G if and only if there is a vertex $v \in V$ such that there exists a nonnegative prefix path from v^1 to v^5 in G_1. In this case, since the reduction is subcubic and the time to check all vertices in G_1 is $O(n)$, an $O(n^{3-\varepsilon})$ time algorithm for the All-Pairs Nonnegative Prefix Paths problem would imply an $O(n^{3-\varepsilon})$ time algorithm for the Negative Triangle problem, for any $\varepsilon > 0$, contradicting the APSP Hypothesis.

We proceed with the proof of the claim. Suppose that there are three vertices a, b, c that form a negative weight cycle C in G, and let G_2 be the graph G after flipping the sign of the weights. Then we have that $w_{G_2}(C) > 0$ in G_2, and based on Lemma 3.5 there is a vertex $v \in C$, such that there exists a nonnegative prefix path in G_2 from v to itself along C. Notice that v can be either a, b or c, and by construction, the paths $a^1b^2c^3a^4a^5$, $b^1c^2a^3b^4b^5$, and $c^1a^2b^3c^4c^5$ exist in G_1. Thus without loss of generality, we can assume that v is a and we use the path $a^1b^2c^3a^4a^5$ in G_1. By construction, it holds that $w_{G_1}(a^1, b^2) = w_{G_2}(a,b), w_{G_1}(b^2, c^3) = w_{G_2}(b,c)$, $w_{G_1}(c^3, a^4) = w_{G_2}(c,a)$ and $w_{G_1}(a^4, a^5) = w_{\min}$. The path $abca$ is a nonnegative prefix path in G_2, and so the path $a^1b^2c^3a^4$ is a nonnegative prefix path in G_1 as well. Moreover since $w_{G_2}(C) > 0$, we have that $w_{G_2}(C) \geq -w_{\min}$, which implies that:

$$w_{G_1}(a^1, b^2) + w_{G_1}(b^2, c^3) + w_{G_1}(c^3, a^4) \geq -w_{\min}.$$

Thus, we can conclude that the path $a^1b^2c^3a^4a^5$ is a nonnegative prefix path in G_1.

For the other direction, let $a^1b^2c^3a^4a^5$ be a nonnegative prefix path in G_1. By construction of G_1 and the fact that G does not contain self-loops, it must be the case that the corresponding vertices a, b, c must be pairwise different in G. By definition of a nonnegative prefix path, it holds that:

$$w_{G_1}(a^1, b^2) + w_{G_1}(b^2, c^3) + w_{G_1}(c^3, a^4) \geq -w_{\min} > 0.$$

By construction, we have that $w(a,b) = -w_{G_1}(a^1, b^2), w(b,c) = -w_{G_1}(b^2, c^3)$ and $w(c,a) = -w_{G_1}(c^3, a^4)$. Therefore, it is true that $w(a,b) + w(b,c) + w(c,a) < 0$, and the vertices a, b, c form a negative weight cycle in G. □

4 The All-Alice Case

In this section, we develop an algorithm that computes the minimum sufficient energies of all vertices for game graphs controlled by Alice. In particular, we obtain the following result.

Theorem 1.3. *There exists a deterministic algorithm that, given a game graph $G = (V,E,w)$ in which all vertices are controlled by Alice, computes the minimum sufficient energy of all vertices in $\tilde{O}(n^{\omega}W^{\omega})$ time.*

The idea is to use the algorithm of Theorem 1.2 for the All-Pairs Nonnegative Prefix Paths problem. Hélouët, Markey, and Raha [22] provide a relevant reduction from the problem of whether zero energy suffices to the problem of whether there exists a nonnegative prefix path. Hence, one idea would be to apply this reduction and run the algorithm of Theorem 1.2. Unfortunately this reduction affects the weights, and the maximum weight of the new instance can be as big as mW, which in turn affects the running time of the algorithm.

To that end, we present another way to use the All-Pairs Nonnegative Prefix Paths algorithm of Theorem 1.2 without affecting the maximum weight of the graph. The algorithm consists of two phases. In the first phase, we detect all the vertices such that initial zero energy suffices, and in the second phase we compute the minimum sufficient energy for the rest of the vertices.

In the first phase of the algorithm, initially we run the All-Pairs Nonnegative Prefix Paths algorithm of Theorem 1.2 on the game graph $G = (V,E,w)$. Hence, we retrieve the information of whether there exists a nonnegative prefix path from a vertex u to a vertex v, for any two vertices $u,v \in V \times V$. Then for each vertex $v \in V$, we check whether there is a vertex u (including v) such that there exists a nonnegative prefix path from v to u and from u to u. If this is the case, then we add this vertex to a set Z. The next lemma shows that the set Z is actually the set of all vertices such that initial energy zero suffices.

Lemma 4.1. *The set Z is the same as the set $\{v \in V : e^*(v) = 0\}$, and is computed in $\tilde{O}(n^{\omega}W^{\omega})$ time.*

Proof. Suppose that the algorithm adds a vertex v to Z. Then, there must be a vertex u (possibly $u = v$) such that there exists a nonnegative prefix path from v to u and from u to u. By merging then these two paths, and by definition of minimum sufficient energy, we can conclude that $e^*(v) = 0$.

Suppose now that the minimum sufficient energy of a vertex $v \in V$ is zero (i.e., $e^*(v) = 0$). By definition of minimum sufficient energy, there must exist a nonnegative prefix lasso P which contains a nonnegative cycle C. Also by Lemma 3.5, there is a vertex $u \in C$ in the cycle, such that there exists a nonnegative prefix path from u to itself. As a result, the algorithm finds these vertices v and u and adds v to Z.

The running time of this process is dominated by the running time of the All-Pairs Nonnegative Prefix Paths algorithm, which is $\tilde{O}(n^{\omega}W^{\omega})$ based on Theorem 1.2. □

The set Z can be seen as the set of possible vertices to 'end' in. Any optimal strategy would still have to define how to move from such a vertex $v \in Z$, but since we know that $e^*(v) = 0$, there has to be a path such that from this vertex no initial energy is necessary. So the goal of the second phase, is to find for each vertex $v \in V \setminus Z$ the best way to hit a vertex in Z. The following lemma shows that this comes down to a shortest path computation. Brim and Chaloupka [8] use a similar idea inside their subroutine for the Mean-Payoff games.

Lemma 4.2. *Given a game graph $G = (V,E,w)$ where all vertices belong to Alice and the set $Z := \{v \in V : e^*(v) = 0\}$ is known, we can compute the remaining minimum sufficient energies through a single SSSP computation in G.*

For the proof we refer to the full version of the paper. Together, Lemma 4.1 and Lemma 4.2 prove Theorem 1.3, by using also the fact that we can compute SSSP deterministically in $\tilde{O}(n^{\omega}W)$ time [27, 30].

5 The All-Bob Case

In this section, we restrict ourselves to the case where all vertices belong to Bob. We show that this special case admits a near-linear time algorithm, by essentially reducing the problem to detecting negative cycles and computing shortest paths. We obtain the following result.

Theorem 1.4. *There exists a randomized (Las Vegas) algorithm that, given a game graph $G = (V,E,w)$ in which all vertices are controlled by Bob, computes the minimum sufficient energy of all vertices, and with high probability the algorithm takes $O(m \log^2 n \log nW \log\log n)$ time.*

Proof. We split the algorithm and proof in two parts, depending on who wins the game in a particular vertex. The first part of the algorithm consists of identifying the vertices with infinite energy (namely, the vertices where Bob wins), and the second part consists of calculating the finite energies of the remaining vertices (namely, the vertices where Bob loses).

Vertices where Bob wins. First, we identify the vertices where Bob wins, i.e., the vertices v with $e^*(v) = \infty$. Hereto, we decompose G in to strongly connected components $C_1, \ldots, C_r$, for some $r \geq 1$. On each C_i, we run a negative cycle detection algorithm. If there is a negative cycle, we set $e(v) = \infty$ for all $v \in C_i$. Next we find the vertices that can reach these cycles. Let $A := \{v \in V : e(v) = \infty\}$ be the union of the strongly connected components with a negative cycle. Then from A we run an inward reachability algorithm (e.g., DFS, BFS) towards each vertex v and if there is a path from v to A, we set $e(v) = \infty$. In the correctness proof, we show that $e(v) = \infty$ if and only if Bob wins at v.

Correctness. For any vertex $v \in V$, Bob wins if and only if there is a path from v to a negative cycle. Let v be a vertex where Bob wins, and let $C^{(v)}$ be the negative cycle reachable from v. If v belongs to the strongly connected component of $C^{(v)}$, then our algorithm outputs $e(v) = \infty$. If v belongs to a different connected component, then the path to the negative cycle is detected in the inward reachability algorithm and we also output $e(v) = \infty$.

Suppose we output $e(v) = \infty$. If we do this because v belongs to a strongly connected in which we detected a negative cycle, then clearly there is path from v to the negative cycle, and hence Bob wins at v. If we set $e(v) = \infty$ because there is a path from v to A, then there is a path from v towards a strongly connected component containing a negative cycle, and hence to a negative cycle itself. So again Bob wins at v.

Running time. We can decompose G in to strongly connected components in $O(m)$ time [28]. On each connected component C_i, we can detect whether there is a negative cycle in the graph in $O(|E(C_i)| \log^2 n \log nW \log\log n)$ time w.h.p. [10], thus the total time is $O(m \log^2 n \log nW \log\log n)$ w.h.p. The inward reachability algorithm can be implemented by a simple DFS or BFS in $O(m)$ time. Hence in total we obtain w.h.p a running time of $O(m \log^2 n \log nW \log\log n)$ for this part.

Vertices where Bob loses. Second, we compute the correct value for the vertices where Bob loses, i.e., the vertices v with $e(v) < \infty$. Note that for this part we can restrict ourselves to the subgraph where we omit all vertices with $e(v) = \infty$. We also add a new sink vertex t to the graph, and for every $v \in V$ we insert an edge (v,t) with $w(v,t) = 0$. Now for each vertex v, we compute the minimum distance $d(v,t)$ from v to t, and we set $e(v) = \max\{-d(v,t), 0\}$. In the correctness proof, we show that $e^*(v) = e(v)$ for each $v \in V$ with $e(v) < \infty$.

Correctness. Consider now a vertex v such that $e(v) < \infty$. First we show that $e^*(v) \geq e(v)$. Let u be the last vertex (excluding t itself) on the shortest path from v to t, and $P_{v,u}$ be the corresponding prefix from v to u. Then Bob can choose to move along the path $P_{v,u}$ forcing Alice to use at least $\max\{-w(P_{v,u}), 0\}$ initial energy. As $d(v,t) = w(P_{v,u}) + w(u,t) = w(P_{v,u}) + 0 = w(P_{v,u})$, we conclude that Alice needs at least $\max\{-d(v,t), 0\} = e(v)$ initial energy.

It remains to show $e^*(v) \leq e(v)$. Since there are no negative cycles, by definition we have that $e^*(v) = \max\{-\min_{u \in V} w(P_u), 0\}$, where the minimization is over all the simple paths from v to u. Also for all $u \in V$, it holds that $d(v,u) \leq w(P_u)$ and $d(v,t) \leq d(v,u) + w(u,t) = d(v,u) + 0 = d(v,u)$. Thus we get that $e^*(v) = \max\{-\min_{u \in V} w(P_u), 0\} \leq \max\{-\min_{u \in V} d(v,t), 0\} = \max\{-d(v,t), 0\} = e(v)$.

Running time. To compute the shortest paths from v to t, we flip the direction of all the edges and we compute the minimum distances from t to v in the new graph. This clearly corresponds to the minimum distances from v to t in the original graph. Since this computation is the negative weight single source shortest path problem, it can be done in $O(m \log^2 n \log nW \log\log n)$ time w.h.p. [10]. □

6 Game Graphs Without Negative Cycles

In this section, we provide an $O(mn)$ time algorithm for the special case where the game graph has no negative cycles. We do this in three steps: first, we introduce a finite duration energy game, where a token is passed for i rounds. The goal is to compute for each vertex, the minimum initial energy that Alice needs in order to keep the energy nonnegative for those i rounds. Second, we provide an algorithm that computes this value in $O(mi)$ time. Finally, we show that for graphs with no negative cycles, it suffices to find this minimum initial energy for a game of n rounds.

6.1 Finite Duration Games

We introduce a version of the energy game that lasts i rounds. We define strategies and energy functions analogous to the infinite duration game, as in Section 2. A strategy for Alice is a function $\sigma_i : V^* V_A \to V$, such that for all finite paths $u_0 u_1 \cdots u_j$ with $j < i$ and $u_j \in V_A$, we have that $\sigma_i(u_0 u_1 \cdots u_j) = v$ for some edge $(u_j, v) \in E$. Similarly we define a strategy τ_i for Bob by replacing V_A with V_B. A path $u_0 u_1 \cdots u_j$ of length j is consistent with respect to strategies σ_i and τ_i, if $\sigma_i(u_0 u_1 \cdots u_k) = u_{k+1}$ for all $u_k \in V_A$ and $\tau_i(u_0 u_1 \cdots u_k) = u_{k+1}$ for all $u_k \in V_B$, where $0 \leq k < j \leq i$. The minimum sufficient energy at a vertex u corresponding to strategies σ_i and τ_i is defined as $e_{\sigma_i,\tau_i}(u) := \max\{-\min w(P), 0\}$, where the minimization is over all the consistent paths P with respect to σ_i and τ_i of length at most i originating at u. The minimum sufficient energy at a vertex u is defined as follows:

$$e_i^*(u) := \min_{\sigma_i} \max_{\tau_i} e_{\sigma_i,\tau_i}(u),$$

where we minimize over all strategies σ_i for Alice and maximize over all strategies τ_i for Bob. As for the infinite duration game, we know by Martin's determinacy theorem [26] that $\min_{\sigma_i} \max_{\tau_i} e_{\sigma_i,\tau_i}(u) = \max_{\tau_i} \min_{\sigma_i} e_{\sigma_i,\tau_i}(u)$. Now we define *optimal strategies* as follows. A strategy σ_i^* is an optimal strategy for Alice at a vertex u, if for any strategy τ_i for Bob it holds that $e_{\sigma_i^*,\tau_i}(u) \leq e_i^*(u)$. Likewise a strategy τ_i^* is an optimal strategy for Bob at a vertex u, if for any strategy σ_i for Alice it holds that $e_{\sigma_i,\tau_i^*}(u) \geq e_i^*(u)$. A value $e(u)$ is a sufficient energy at a vertex u, if there exists a strategy σ_i such that for any strategy τ_i, it holds that $e_{\sigma_i,\tau_i}(u) \leq e(u)$. In this case, observe that the following is true:

$$e_i^*(u) = \max_{\tau_i} e_{\sigma_i^*,\tau_i}(u) \leq \max_{\tau_i} e_{\sigma_i,\tau_i}(u) \leq e(u).$$

Next, we show the following lemma about the minimum energy function, a similar version has also been used for the infinite duration game in [9] and [15]. For the proof, see the full version of the paper.

Lemma 6.1. *Given a game of i rounds and a vertex $u \in V$, the energy $e_i^*(u)$ satisfies the following properties:*

$$\text{if } u \in V_A \text{ then } \exists v \in N^+(u) : e_i^*(u) + w(u,v) \geq e_{i-1}^*(v) \tag{1}$$

$$\text{if } u \in V_B \text{ then } \forall v \in N^+(u) : e_i^*(u) + w(u,v) \geq e_{i-1}^*(v) \tag{2}$$

6.2 A Value Iteration Algorithm for Finite Duration Games

In this section, we present Algorithm 1, a value iteration algorithm for a game lasting i rounds that computes for each vertex $u \in V$ the value $e_i^*(u)$. We note that Algorithm 1 consists of i steps, where at every step each edge is scanned at most once. Clearly this means the algorithm takes $O(mi)$ time.

Algorithm 1 Value iteration algorithm for an i-round game

Input: A game graph $G = (V, E, w, \langle V_A, V_B \rangle)$, a number of iterations i

Output: The minimum sufficient energy $e_i(u)$ of each $u \in V$, in order to play the game for i rounds

```
1  $\forall u \in V : e_0(u) \leftarrow 0$
2  for $j = 1$ to $i$ do
3      foreach $u \in V$ do
4          if $u \in V_A$ then
5              $e_j(u) \leftarrow \max\{\min_{(u,v)\in E}\{e_{j-1}(v) - w(u,v)\}, 0\}$
6          end
7          if $u \in V_B$ then
8              $e_j(u) \leftarrow \max\{\max_{(u,v)\in E}\{e_{j-1}(v) - w(u,v)\}, 0\}$
9          end
10     end
11 end
12 return $e_i$
```

Lemma 6.2. *Let $e_i(\cdot)$ be the function returned by Algorithm 1, then $e_i(u) = e_i^*(u)$ for all $u \in V$.*

Proof. We prove the claim by induction on i, which is both the number of steps of the algorithm and the duration of the game.

Base case: For $i = 0$ steps, the algorithm sets for each $u \in V : e_0(u) = 0 = e_0^*(u)$.

Inductive Step: We assume that after $i-1$ steps $e_{i-1}(u) = e_{i-1}^*(u)$, and we prove that after i steps $e_i(u) = e_i^*(u)$ as well. We first show that $e_i(u) \geq e_i^*(u)$.

Consider the case that $u \in V_A$. Let v' be the neighbor that minimizes the relation in the i_{th} step in Line 5. Then it holds that $e_i(u) + w(u,v') \geq e_{i-1}(v')$. Using the edge (u,v') with initial energy $e_i(u)$, Alice can move to v' with remaining energy at least $e_{i-1}(v')$. By the inductive hypothesis it holds that $e_{i-1}(v') = e_{i-1}^*(v')$, so there exists an optimal strategy σ_{i-1}^* such that for any strategy τ_{i-1}, we have that $e_{\sigma_{i-1}^*,\tau_{i-1}}(v') \leq e_{i-1}(v')$. Define the strategy σ_i in the following way: $\forall x \in V^*V_A : \sigma_i(ux) = \sigma_{i-1}^*(x)$ and $\sigma_i(u) = v'$. Then we get a strategy σ_i such that for any strategy τ_i, it holds that $e_{\sigma_i,\tau_i}(u) \leq e_i(u)$. This implies that $e_i(u)$ is a sufficient energy at vertex u, and so $e_i(u) \geq e_i^*(u)$.

Consider the case that $u \in V_B$. Due to the i_{th} step in Line 8, it holds that $e_i(u) + w(u,v) \geq e_{i-1}(v)$, for all $v \in N^+(u)$. Hence for any choice of a neighboring edge (u,v) with initial energy $e_i(u)$, Bob moves to a neighbor v with remaining energy at least $e_{i-1}(v)$. By the inductive hypothesis, for all $v \in N^+(u)$ it holds that $e_{i-1}(v) = e^*_{i-1}(v)$, so there exists an optimal strategy σ^*_{i-1} such that for any strategy τ_{i-1}, we have that $e_{\sigma^*_{i-1},\tau_{i-1}}(v) \le e_{i-1}(v)$. Define the strategy σ_i in the following way: $\forall x \in V^*V_A : \sigma_i(ux) = \sigma^*_{i-1}(x)$. Then we get a strategy σ_i such that for any strategy τ_i, it holds that $e_{\sigma_i,\tau_i}(u) \le e_i(u)$. This implies that $e_i(u)$ is a sufficient energy at vertex u, and so $e_i(u) \ge e^*_i(u)$.

It remains to show that $e_i(u) \le e^*_i(u)$. Consider the case that $u \in V_A$. If $e_i(u) = 0$ then the claim holds trivially. If $e_i(u) > 0$, then based on Line 5, we have that $e_i(u) + w(u,v) \le e_{i-1}(v)$ for all $v \in N^+(u)$. By Lemma 6.1, there exists $v' \in N^+(u)$ such that $e^*_i(u) + w(u,v') \ge e^*_{i-1}(v')$, which means that:

$$e_i(u) + w(u,v') \le e_{i-1}(v') = e^*_{i-1}(v') \le e^*_i(u) + w(u,v') \Rightarrow e_i(u) \le e^*_i(u),$$

where the equality holds by the inductive hypothesis.

Consider the case that $u \in V_B$. If $e_i(u) = 0$ then the claim holds trivially. Otherwise based on Line 8, there exists $v' \in N^+(u)$ such that $e_i(u) + w(u,v') = e_{i-1}(v')$. By Lemma 6.1, we have that $e^*_i(u) + w(u,v) \ge e^*_{i-1}(v)$ for all $v \in N^+(u)$, which means that:

$$e_i(u) + w(u,v') = e_{i-1}(v') = e^*_{i-1}(v') \le e^*_i(u) + w(u,v') \Rightarrow e_i(u) \le e^*_i(u),$$

where the equality holds by the inductive hypothesis. □

6.3 No Negative Cycles

The goal of this section is to show that for graphs with no negative cycles, it holds that $e^*_n(u) = e^*(u)$, for all $u \in V$. Hereto, we show in Lemma 6.4 that as in the infinite duration game, positional strategies suffice when no negative cycles are present. In the proof, we use the following alternative characterization of $e_{\sigma_i,\tau_i}(u)$.

Let σ_i and τ_i be strategies for Alice and Bob respectively, and let $u \in V$ be a vertex. Moreover, let $u_0u_1 \cdots u_j$ be the consistent path of length j with respect to σ_i and τ_i, where $u_0 = u$. Then given an initial energy e_{init}, the energy level at vertex u_j is equal to the value $e_{\text{init}} + \sum_{k=0}^{j-1} w(u_k, u_{k+1})$. We denote $e^*_{\text{init}}(u)$ for the minimum nonnegative initial energy such that the energy level at each vertex of the corresponding consistent path of length i, is nonnegative. The following lemma shows that $e_{\sigma_i,\tau_i}(u) = e^*_{\text{init}}(u)$ (for the proof, see the full version of the paper).

Lemma 6.3. *For a vertex u and two fixed strategies σ_i and τ_i, let P be the consistent path with respect to σ_i and τ_i of length i originating at u. Then it holds that $e_{\sigma_i,\tau_i}(u) = e^*_{\text{init}}(u)$.*

Now we are ready to show that positional strategies suffice in graphs without negative cycles. For the proof see the full version of the paper.

Lemma 6.4. *Consider a graph with no negative cycles and a game of i rounds. Then for the minimum sufficient energy $e^*_i(u)$ at a vertex $u \in V$, it suffices for both players to play positional strategies.*

We use this fact to show that a game of n rounds is equivalent to a game of infinite duration for a game graph without negative cycles.

Lemma 6.5. *Consider a graph with no negative cycles. Then for each vertex $u \in V$, the minimum sufficient energy needed at u for a game of n rounds, is equal to the minimum sufficient energy needed at u for a game of infinite rounds. In other words, $e^*_n(u) = e^*_\infty(u) = e^*(u)$ for all $u \in V$.*

Proof. Let σ and τ be two arbitrary positional strategies for the infinite duration game. By definition, we have that $e_{\sigma,\tau}(u) = \max\{-\min w(P), 0\}$, where the minimization is over all the consistent paths with respect to σ and τ originating at u. Since the graph contains only nonnegative cycles and the strategies are positional, the path that minimizes the relation is a simple path, and so, its length is at most n. Hence it follows that $e_{\sigma,\tau}(u) = \max\{-\min_{|P|\le n} w(P), 0\}$. In turn, this is equivalent to using positional strategies for a game of n rounds. Hence it holds that $e_{\sigma,\tau}(u) = e_{\sigma_n,\tau_n}(u)$, where σ_n and τ_n are the strategies σ and τ respectively, restricted to the first n rounds. This implies that $e^*(u) = \min_{\sigma_n} \max_{\tau_n} e_{\sigma_n,\tau_n}(u)$, where σ_n and τ_n are positional strategies for a game of n rounds. By Lemma 6.4, this equals $e^*_n(u)$ and the claim follows. ☐

Together, Lemma 6.2 and Lemma 6.5 prove Theorem 1.5.

References

[1] Alfred V. Aho, John E. Hopcroft & Jeffrey D. Ullman (1974): *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

[2] Noga Alon, Zvi Galil & Oded Margalit (1997): *On the Exponent of the All Pairs Shortest Path Problem*. *J. Comput. Syst. Sci.* 54(2), pp. 255–262, doi:10.1006/jcss.1997.1388.

[3] Aaron Bernstein, Danupon Nanongkai & Christian Wulff-Nilsen (2022): *Negative-Weight Single-Source Shortest Paths in Near-linear Time*. In: *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, IEEE, pp. 600–611, doi:10.1109/FOCS54457.2022.00063.

[4] Henrik Björklund & Sergei G. Vorobyov (2007): *A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games*. *Discrete Applied Mathematics* 155(2), pp. 210–229, doi:10.1016/j.dam.2006.04.029. Announced at MFCS 2004.

[5] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger & Barbara Jobstmann (2009): *Better Quality in Synthesis through Quantitative Objectives*. In: *Proc. of the 21st International Conference on Computer Aided Verification (CAV 2009)*, *Lecture Notes in Computer Science* 5643, Springer, pp. 140–156, doi:10.1007/978-3-642-02658-4_14. arXiv:0904.2638.

[6] Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey & Jirí Srba (2008): *Infinite Runs in Weighted Timed Automata with Energy Constraints*. In Franck Cassez & Claude Jard, editors: *Proc. of the 6th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2008)*, *Lecture Notes in Computer Science* 5215, Springer, pp. 33–47, doi:10.1007/978-3-540-85778-5_4.

[7] Phillip G. Bradford (2017): *Efficient exact paths for dyck and semi-dyck labeled path reachability (extended abstract)*. In: *Proc. of the 8th IEEE Annual Conference on Ubiquitous Computing, Electronics and Mobile Communication (UEMCON 2017)*, IEEE, pp. 247–253, doi:10.1109/UEMCON.2017.8249039.

[8] Luboš Brim & Jakub Chaloupka (2012): *Using strategy improvement to stay alive*. *International Journal of Foundations of Computer Science* 23(03), pp. 585–608, doi:10.1142/S0129054112400291.

[9] Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini & Jean-François Raskin (2011): *Faster algorithms for mean-payoff games*. *Formal Methods in System Design* 38(2), pp. 97–118, doi:10.1007/s10703-010-0105-x. Announced at MEMICS 2009 and GAMES 2009.

[10] Karl Bringmann, Alejandro Cassis & Nick Fischer (2023): *Negative-Weight Single-Source Shortest Paths in Near-Linear Time: Now Faster!* *arXiv preprint arXiv:2304.05279*, doi:10.48550/arXiv.2304.05279.

[11] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li & Frank Stephan (2022): *Deciding Parity Games in Quasi-polynomial Time*. *SIAM Journal on Computing* 51(2), pp. 17–152, doi:10.1137/17m1145288. Announced at STOC 2017.

[12] Pavol Cerný, Krishnendu Chatterjee, Thomas A. Henzinger, Arjun Radhakrishna & Rohit Singh (2011): *Quantitative Synthesis for Concurrent Programs*. In: *Proc. of the 23rd International Conference on Computer Aided Verification (CAV 2011)*, *Lecture Notes in Computer Science* 6806, Springer, pp. 243–259, doi:10.1007/978-3-642-22110-1_20. arXiv:1104.4306.

[13] Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger & Mariëlle Stoelinga (2003): *Resource Interfaces*. In Rajeev Alur & Insup Lee, editors: *Proc. of the Third International Conference on Embedded Software (EMSOFT 2003)*, *Lecture Notes in Computer Science* 2855, Springer, pp. 117–133, doi:10.1007/978-3-540-45212-6_9.

[14] Krishnendu Chatterjee, Laurent Doyen, Mickael Randour & Jean-François Raskin (2015): *Looking at mean-payoff and total-payoff through windows*. *Inf. Comput.* 242, pp. 25–52, doi:10.1016/j.ic.2015.03.010.

[15] Krishnendu Chatterjee, Monika Henzinger, Sebastian Krinninger & Danupon Nanongkai (2014): *Polynomial-Time Algorithms for Energy Games with Special Weight Structures*. *Algorithmica* 70(3), pp. 457–492, doi:10.1007/s00453-013-9843-7. arXiv:1604.08234. Announced at ESA 2012.

[16] Carlo Comin & Romeo Rizzi (2017): *Improved Pseudo-polynomial Bound for the Value Problem and Optimal Strategy Synthesis in Mean Payoff Games*. *Algorithmica* 77(4), pp. 995–1021, doi:10.1007/s00453-016-0123-1.

[17] Dani Dorfman, Haim Kaplan, Robert E. Tarjan & Uri Zwick (2023): *Optimal Energetic Paths for Electric Cars*. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi & Grzegorz Herman, editors: *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, *LIPIcs* 274, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 42:1–42:17, doi:10.4230/LIPIcs.ESA.2023.42.

[18] Dani Dorfman, Haim Kaplan & Uri Zwick (2019): *A Faster Deterministic Exponential Time Algorithm for Energy Games and Mean Payoff Games*. In: *Proc. of the 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, 132, pp. 114:1–114:14, doi:10.4230/LIPIcs.ICALP.2019.114.

[19] Andrzej Ehrenfeucht & Jan Mycielski (1979): *Positional strategies for mean payoff games*. *International Journal of Game Theory* 8(2), pp. 109–113, doi:10.1007/BF01768705.

[20] Nathanaël Fijalkow, Paweł Gawrychowski & Pierre Ohlmann (2020): *Value Iteration Using Universal Graphs and the Complexity of Mean Payoff Games*. In: *Proc. of the 45th International Symposium on Mathematical Foundations of Computer Science (MFCS 2020)*, 170, pp. 34:1–34:15, doi:10.4230/LIPIcs.MFCS.2020.34.

[21] Vladimir A. Gurvich, Alexander V. Karzanov & L. G. Khachivan (1988): *Cyclic games and an algorithm to find minimax cycle means in directed graphs*. *USSR Computational Mathematics and Mathematical Physics* 28(5), pp. 85–91, doi:10.1016/0041-5553(88)90012-2.

[22] Loïc Hélouët, Nicolas Markey & Ritam Raha (2019): *Reachability Games with Relaxed Energy Constraints*. In: *Proceedings Tenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2019, Bordeaux, France, 2-3rd September 2019*, *EPTCS* 305, pp. 17–33, doi:10.4204/EPTCS.305.2.

[23] Donald B. Johnson (1977): *Efficient Algorithms for Shortest Paths in Sparse Networks*. *J. ACM* 24(1), pp. 1–13, doi:10.1145/321992.321993.

[24] Marcin Jurdziński (1998): *Deciding the winner in parity games is in UP$\cap$ co-UP*. *Information Processing Letters* 68(3), pp. 119–124, doi:10.1016/S0020-0190(98)00150-1.

[25] Tomasz Kociumaka & Adam Polak (2023): *Bellman-Ford Is Optimal for Shortest Hop-Bounded Paths*. In Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi & Grzegorz Herman, editors: *31st Annual European Symposium on Algorithms, ESA 2023, September 4-6, 2023, Amsterdam, The Netherlands*, *LIPIcs* 274, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 72:1–72:10, doi:10.4230/LIPIcs.ESA.2023.72. arXiv:2211.07325.

[26] Donald A. Martin (1975): *Borel determinacy*. *Annals of Mathematics* 102(2), pp. 363–371, doi:10.2307/1971035.

[27] Piotr Sankowski (2005): *Shortest Paths in Matrix Multiplication Time*. In Gerth Stølting Brodal & Stefano Leonardi, editors: *Algorithms - ESA 2005, 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005, Proceedings*, *Lecture Notes in Computer Science* 3669, Springer, pp. 770–778, doi:10.1007/11561071_68.

[28] Robert E. Tarjan (1972): *Depth-First Search and Linear Graph Algorithms*. *SIAM J. Comput.* 1(2), pp. 146–160, doi:10.1137/0201010.

[29] Virginia Vassilevska Williams & R. Ryan Williams (2018): *Subcubic Equivalences Between Path, Matrix, and Triangle Problems*. *J. ACM* 65(5), pp. 27:1–27:38, doi:10.1145/3186893. Announced at FOCS 2010.

[30] Raphael Yuster & Uri Zwick (2005): *Answering distance queries in directed graphs using fast matrix multiplication*. In: *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings*, IEEE Computer Society, pp. 389–396, doi:10.1109/SFCS.2005.20.

[31] Uri Zwick & Mike Paterson (1996): *The complexity of mean payoff games on graphs*. *Theoretical Computer Science* 158(1-2), pp. 343–359, doi:10.1016/0304-3975(95)00188-3.